UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROGRAMA DE EDUCAÇÃO TUTORIAL

Apostila de MATLAB

Fortaleza – CE

Maio / 2014

Responsáveis

A apostila de **MATLAB** é de responsabilidade do **Programa de Educação Tutorial** do curso de **Engenharia Elétrica** da **Universidade Federal do Ceará**, tendo como principais responsáveis desta versão os bolsistas:

AUTORES:

- **Decio Haramura Junior**
- **Guilherme Martins Gomes Nascimento**
- **Luís Paulo Carvalho dos Santos**
- **Luís Fernando Almeida Fontenele**
- **Pedro André Martins Bezerra**

CO-AUTORES:

- **Abnadan de Melo Martins**
- **Francisco Onivaldo de Oliveira Segundo**
- **Janailson Rodrigues Lima**
- **Lucas Chaves Gurgel**
- **Raphael Fernandes Sales Costa**

REVISORES:

- **Alexcya Lopes Alexandre**
- **Camila Tavares Vitoriano**
- **Felipe Carvalho Sampaio**
- **Ícaro Silvestre Freitas Gomes**
- **José Antonio de Barros Filho**
- **Lucas Cordeiro Herculano**
- **Lucas Rebouças Maia**
- **Maria Yasmin Almeida Sampaio**
- **Nestor Rocha Monte Fontenele**
- **Ricardo Antônio de Oliveira Sousa Junior**
- **Roberto Aaron Marques Braga**
- **Túlio Naamã Guimarães Oliveira**

SUMÁRIO

1. PREFÁCIO

Esta apostila foi desenvolvida por alunos do Programa de Educação Tutorial (PET) do curso de Engenharia Elétrica da Universidade Federal do Ceará (UFC) para a realização do Curso de MATLAB.

Com o intuito de promover uma introdução ao MATLAB que viesse a facilitar o desempenho dos estudantes da graduação na realização de seus trabalhos e na sua vida profissional, o PET elaborou este Curso de MATLAB que está atualmente na décima quinta edição. Durante as quinze edições foram contemplados aproximadamente 700 estudantes dos mais variados cursos de Engenharia do Centro de Tecnologia da UFC.

Devido à sua boa repercussão o Curso de MATLAB foi premiado no XVII Encontro de Iniciação à Docência nos Encontros Universitários de 2008.

2. APRESENTAÇÃO

O MATLAB (MATrix LABoratory) é uma linguagem de alto desempenho para computação técnica. Integra computação, visualização e programação em um ambiente de fácil uso onde problemas e soluções são expressos em linguagem matemática. Usos típicos:

- Matemática e computação;
- Desenvolvimento de algoritmos;
- Aquisição de dados;
- Modelagem, simulação e prototipagem;
- Análise de dados, exploração e visualização;
- Construção de interface visual do usuário.

2.1. Utilizando o HELP

Indubitavelmente, a melhor apostila tutorial sobre o MATLAB que possa existir é o *HELP* do próprio MATLAB. Todas as informações possíveis há no *HELP*, principalmente sobre as *toolboxes*, sobre funções, *SIMULINK* e entre outros.

O *HELP* pode ser aberto de várias formas. A primeira é através da barra de menu, como mostrado na Fig. 1.

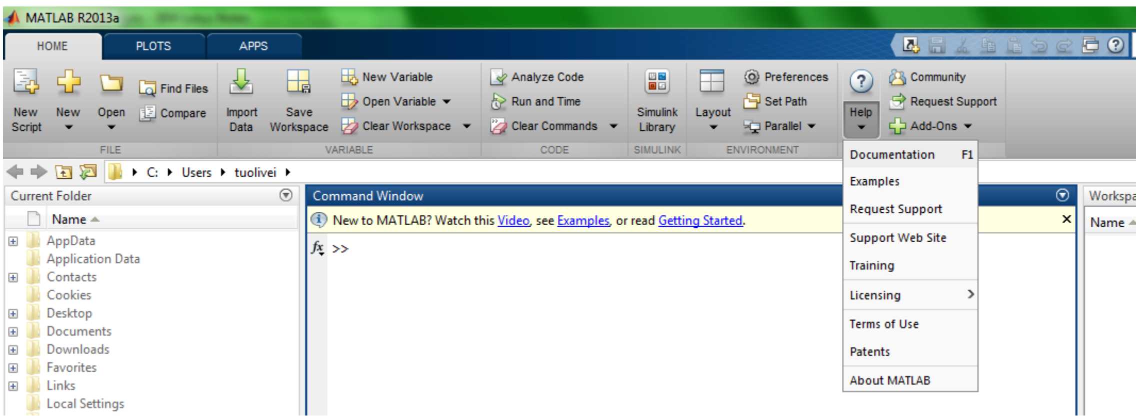


Figura 1 - *HELP* do MATLAB sendo acessado pela barra de menu

Outra forma é pela tecla de atalho F1. Uma terceira forma é pelo botão da barra de acesso rápido, posicionado logo acima da barra de Menu, de acordo com a Fig. 2.

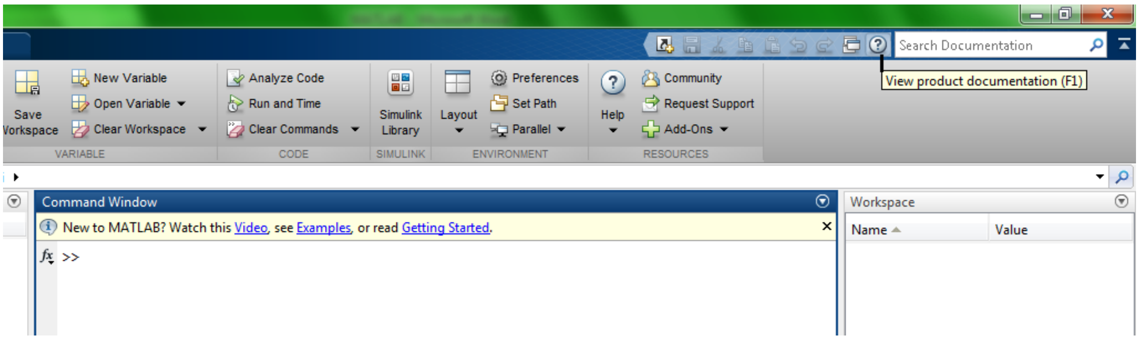


Figura 2 - *HELP* do MATLAB sendo acessado pelo botão *START*

Dando continuidade, quando se deseja obter informações sobre uma dada função, é possível consultar diretamente no *HELP* ou pelo *COMMAND WINDOW*. Para isso, basta digitar *help* e em seguida a função requerida, de acordo com o exemplo abaixo:

```
>> help dirac
 dirac  Delta function.
     dirac(X) is zero for all X, except X == 0 where it is
     infinite.
     dirac(X) is not a function in the strict sense, but rather a
     distribution with int(dirac(x-a)*f(x),-inf,inf) = f(a) and
     diff(heaviside(x),x) = dirac(x).
     See also heaviside.
```

Veja que as informações sobre a função *dirac* aparecem no próprio *COMMAND WINDOW*. Se for necessário consultar a página do *HELP*, basta utilizar o comando *doc* e em seguida o nome da função. Por exemplo:

```
>> doc dirac
```

Depois de efetuado este comando, irá aparecer a janela do *HELP* com o seguinte, Fig. 3.

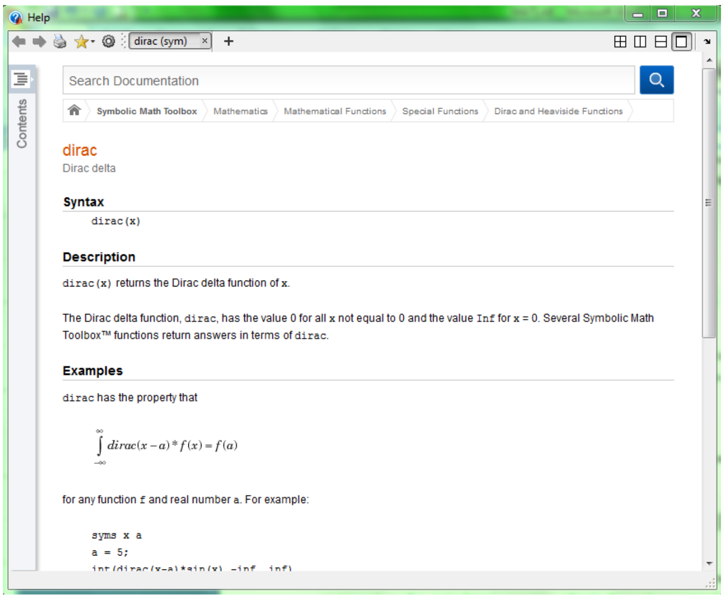


Figura 3 - *HELP* da função *dirac*

3. FORMATAÇÃO

No MATLAB não há necessidade de declarar o tipo das variáveis utilizadas no programa, mas o usuário pode escolher qual o formato que vai ser utilizado. São usados os comandos mostrados na Tabela 1.

Tabela 1 - Formato das variáveis

Comando MATLAB	Variável	Descrição
Format long	3.141592653589793	Com 16 dígitos
Format short	3.1416	Com 5 dígitos
Format short e	3.1416e+000	Com 5 dígitos – notação científica
Format long e	*3.141592653589793e+000*	Com 16 dígitos em notação científica
Format +	+	Retorna “+” para valores positivos e “-” para valores negativos
Format rat	355/113	Aproximação racional
Format hex	400921fb54442d18	Formato hexadecimal

4. MATRIZES

4.1. Declaração

A declaração de matrizes é feita da seguinte maneira:

```
>> a = [1:10]       %cria o vetor linha [1 2 3 4 5 6 7 8 9 10]
>> b = [0:0.5:3]    %cria o vetor [0 0.5 1 1.5 2 2.5 3]
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
%cria a matriz A no seguinte formato
%A =
%    16      3      2     13
%     5     10     11      8
%     9      6      7     12
%     4     15     14      1
>> A(1,2)  %Elemento de linha 1 e coluna 2
>> A(:,3)  %Elementos da coluna 3
>> A(1,:)  %Elementos da linha 1
```

O MATLAB também aceita a concatenação de matrizes, por exemplo:

```
>> a = [4 1; 3 4];
>> b = [2 3; 4 5];
>> c = [a b]
%c =
%     4      1      2      3
%     3      4      4      5
```

Obs.: É bom lembrar que o MATLAB tem como primeiro índice do vetor o número 1, diferente de outras linguagens que usam o primeiro índice como 0.

4.2. Soma

A soma de todos os elementos de uma matriz com um número é feita da seguinte maneira:

```
>> c = [4 1 2 3; 3 4 4 5];
>> c+1
%ans =
%     5      2      3      4
%     4      5      5      6
```

A soma de matrizes é feita da maneira tradicional:

```
>> d = [ 1 2 7 8; 4 7 5 8];
>> e = [ 5 -4 7 0; 3 -1 6 -4];
>> d+e
%ans =
%     6     -2     14      8
%     7      6     11      4
```

4.3. Multiplicação

Usa-se o sinal da multiplicação:

```
>> a = [1 4 2; 7 8 5; 9 5 4];
>> b = [4 2 -5; 0 1 3; 8 -2 1];
>> c = a*b
%c =
%    20      2      9
%    68     12     -6
%    68     15    -26
```

Obs.: Se for desejado realizar outra operação matemática (exceto a soma e a subtração) entre os elementos com mesmo índice das matrizes deve-se colocar um ponto antes do operador. Observe os exemplos abaixo:

```
>> a = [1 4 2; 7 8 5; 9 5 4];
>> b = [4 2 -5; 0 1 3; 8 -2 1];
>> c = a.*b
%c =
%      4      8    -10
%      0      8     15
%     72    -10      4
>> b./a
%ans =
%    4.0000    0.5000   -2.5000
%         0    0.1250    0.6000
%    0.8889   -0.4000    0.2500
>> a.^2
%ans =
%     1     16      4
%    49     64     25
%    81     25     16
```

Exercício 1 - Declare as matrizes A, B e C abaixo:

$$A=\begin{bmatrix}1 & 2 & 3 & 4 & 5 & 6 & 7\end{bmatrix} \qquad B=\begin{bmatrix}3 & 6 & 9 & 12 & 15 & 18 & 21\end{bmatrix}$$

$$C=\begin{bmatrix}0 & 5 & 10 & 15 & 10 & 5 & 0\\ -1 & -2 & -3 & -4 & -5 & -6 & -7\end{bmatrix}$$

Através das matrizes acima, determine as matrizes a seguir utilizando os comandos já mencionados.

$$D=\begin{bmatrix}4 & 5 & 6 & 7\end{bmatrix}$$

$$E=\begin{bmatrix}7 & 6 & 5 & 4\end{bmatrix}$$

$$F=\begin{bmatrix}4 & 5 & 6 & 7\\ 7 & 6 & 5 & 4\end{bmatrix}$$

$$G=\begin{bmatrix}4 & 5\\ 7 & 6\\ 6 & 7\\ 5 & 4\end{bmatrix} \qquad H=\begin{bmatrix}12\\ 15\\ 18\\ 21\end{bmatrix} \qquad I=\begin{bmatrix}5\\ -6\end{bmatrix}$$

$$J=\begin{bmatrix}0 & 5 & 10\\ -1 & -2 & -3\end{bmatrix} \qquad L=\begin{bmatrix}0 & 24 & 99\\ 2 & 5 & 10\end{bmatrix}$$

$$K=\begin{bmatrix}0 & 25 & 100\\ 1 & 4 & 9\end{bmatrix} \qquad M=\begin{bmatrix}0 & -24 & -99\\ -2 & -5 & -10\end{bmatrix}$$

4.4. Matrizes pré-Definidas

- ***ones***

Definição: Esta função gera uma matriz cujos valores são unitários.

Sintaxe:

ones(n) → Gera uma matriz quadrada de ordem *n* cujos termos são unitários.

ones(m,n) → Gera uma matriz *m* x *n* cujos termos são unitários.

```
>> ones(2)
%ans =
%     1     1
%     1     1
```

- ***zeros***

Definição: Esta função gera uma matriz cujos valores são nulos.

Sintaxe:

zeros(n) → Gera uma matriz quadrada de ordem *n* cujos termos são nulos.

zeros(m,n) → Gera uma matriz *m* x *n* cujos termos são nulos.

```
>> zeros(2)
%ans =
%     0     0
%     0     0
```

- ***eye***

Definição: Gera uma matriz identidade.

Sintaxe:

eye(n) → Gera uma matriz identidade *n* x *n*.

eye(m,n) → Gera uma matriz de ordem *m* x *n* cujos termos que possuem i=j são unitários.

```
>> eye(2)
%ans =
%     1     0
%     0     1
```

- ***vander***

Definição: Calcula a matriz de Vandermonde a partir de um vetor dado.

Sintaxe:

vander(A) → Calcula a matriz de Vandermonde a partir de A

```
>> A = [1 2 3 4];
>> vander(A)
%ans =
%     1     1     1     1
%     8     4     2     1
%    27     9     3     1
%    64    16     4     1
```

- ***rand***

Definição: Cria uma matriz com valores aleatórios.

Sintaxe:

rand(m) → Cria uma matriz m x m com valores aleatórios entre 0 e 1.

rand(m,n) → Cria uma matriz m x n com valores aleatórios entre 0 e 1.

```
>> rand(2)
%ans =
%    0.9501    0.6068
%    0.2311    0.4860
```

4.5. Propriedades de matrizes

- **' (apóstrofo)**

Definição: Calcula a matriz transposta.

Sintaxe:

A' → Gera a matriz transposta de A.

```
>> A = [1 1; 2 3];
>> A'
%ans =
%     1     2
%     1     3
```

Definição: Calcula o determinante de uma matriz.

Sintaxe:

det(A) → Calcula o determinante da matriz A.

```
>> A = [1 1; 2 3];
>> det(A)
%ans =
%     1
```

- ***trace***

Definição: Retorna um vetor com a soma dos elementos da diagonal principal de uma matriz.

Sintaxe:

trace(A) → Retorna a soma dos elementos da diagonal principal da matriz A.

```
>> A = [1 4; 2 1];
>> trace(A)
%ans =
%    2
```

- ***inv***

Definição: Determina a matriz inversa dada.

Sintaxe:

inv(A) → Retorna a matriz inversa da matriz A.

```
>> A = [5 8; 4 9];
>> inv(A)
%ans =
%   0.6923   -0.6154
%  -0.3077    0.3846
```

- ***eig***

Definição: Calcula os autovalores e autovetores de uma matriz.

Sintaxe:

eig(A) → Retorna os autovalores de uma matriz quadrada A.

[a, b] = *eig*(A) → Retorna em a uma matriz com os autovetores e, em b, uma matriz com os autovalores associados.

```
>> A = [1 -1; 4 1];
>> [a,b] = eig(A)
%a =
%  0.0000 - 0.4472i   0.0000 + 0.4472i
% -0.8944 + 0.0000i  -0.8944 + 0.0000i
%b =
%  1.0000 + 2.0000i   0.0000 + 0.0000i
%  0.0000 + 0.0000i   1.0000 - 2.0000i
```

Exercício 2 - Resolva o seguinte sistema de equações lineares: $\begin{cases} 2.x_1 + 1{,}5.x_2 + x_3 = 13{,}20 \\ x_1 + 6.x_2 - 2.x_3 = 21{,}64 \\ 2.x_2 + 4.x_3 = 26{,}20 \end{cases}$

4.6. Trabalhando com matrizes

- ***size***

Definição: Retorna as dimensões de uma matriz

Sintaxe:

[m,n] = *size*(A) → Retorna, em m, o número de linhas e, em n, o número de colunas da matriz A.

```
>> A = [1 1; 2 3];
>> [m,n]=size(A)
%m =
%     2
%n =
%     2
```

- ***find***

Definição: Procura os elementos em uma matriz de tal modo a respeitar a lógica fornecida, retornando os índices que descrevem estes elementos.

Sintaxe:

ind = *find*(X) → Retorna os índices de elementos não-nulos na matriz X.

[*row*,*col*] = *find*(X, ...) → Retorna, em *row*, uma matriz coluna com os índices das linhas dos elementos da matriz X e, em *col*, a matriz coluna contendo os índices correspondentes às colunas dos elementos da matriz X.

[*row*,*col*,v] = *find*(X, ...) → Retorna, em *row*, uma matriz coluna com os índices das linhas dos elementos da matriz X , em *col*, uma matriz coluna contendo os índices que descrevem as colunas dos elementos da matriz X e, em v, uma matriz contendo os elementos de X.

```
>> A=[1 0; 0 3];
>> find(A)
%ans =
%     1
%     4
>> X = [3 2 0; -5 0 7; 0 0 1];
>> [r,c,v] = find(X>2);
>> [r c]
%ans =
%     1     1
%     2     3
```

Veja no ultimo caso acima que r e c retornam os índices das linhas e das colunas correspondentes aos elementos que respeitam a expressão oferecida.

Obviamente, os elementos a_{11} e a_{23} são os únicos maiores que 2.

- ***sort***

Definição: Retorna o vetor dado ou elementos de uma matriz em ordem crescente ou decrescente.

Sintaxe:

sort(A,dim) → Retorna os elementos das colunas (dim = 1) ou da linha (dim = 2) da matriz A em ordem crescente.

sort(A,*mode*) → Retorna os elementos das colunas da matriz A em ordem crescente (*mode* = '*ascend*') ou em ordem decrescente (*mode* = '*descend*').

```
>> A=[1 1; 0 3];
>> sort(A)
%ans =
%     0     1
%     1     3
```

- ***fliplr***

Definição: Espelha as colunas de uma matriz.

Sintaxe:

fliplr(A) → Espelha as colunas da matriz A.

```
>> A = [1 2;3 4];
>> fliplr(A)
%ans =
%     2     1
%     4     3
```

- ***flipud***

Definição: Espelha as linhas de uma matriz.

Sintaxe:

flipud(A) → Espelha as linhas da matriz A.

```
>> A = [1 2;3 4];
>> fliplr(A)
%ans =
%     3     4
%     1     2
```

Exercício 3 - Crie um vetor A de 50 elementos aleatórios e em seguida crie a partir deste, outro vetor B obedecendo aos seguintes critérios:

a. Conter somente os elementos de A maiores que 0.5;

b. Os elementos devem de B estar em ordem decrescente.

Exercício 4 - Realize as seguintes operações no MATLAB, a partir das matrizes dadas, e interprete o resultado.

$$A = \begin{pmatrix} 5 & 8 & 6 \\ 9 & 2 & 10 \\ 7 & 6 & 1 \end{pmatrix} \qquad B = \begin{bmatrix} 7 \\ 1 \\ 6 \end{bmatrix} \qquad C = \begin{pmatrix} 5.5 & 8.1 & 4.9 \\ 2.1 & 7.4 & 9.2 \\ 1.3 & 4.5 & 3.8 \end{pmatrix} \qquad D = \begin{bmatrix} 4 & 1 & 0 \end{bmatrix}$$

- **a.** E = *det* (A – λ.I) com λ = 6
- **b.** F = A^{-1} . B
- **c.** A \ B
- **d.** A . F
- **e.** B^T . C
- **f.** D . B A / A

Exemplo 1 - Dado o circuito da Fig. 4, calcule as tensões nos nós 1 e 2.

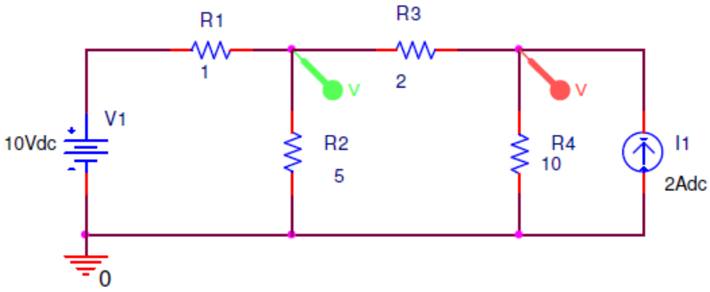


Figura 4 - Exemplo de circuito elétrico

$$i = \frac{1}{R}.v$$

$$i = G\,.v$$

$$\begin{pmatrix} \frac{10}{1} \\ \frac{1}{2} \end{pmatrix} = \begin{pmatrix} \frac{1}{1}+\frac{1}{5}+\frac{1}{2} & \frac{-1}{2} \\ \frac{-1}{2} & \frac{1}{2}+\frac{1}{10} \end{pmatrix} . \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

$$G^{-1}.i = G^{-1}.G.v$$

$$v = G^{-1}.i$$

```
>> i = [10/1; 2];
>> G=[1/1+1/5+1/2 -1/2; -1/2 1/2+1/10]
%G =
%   1.7000   -0.5000
%  -0.5000    0.6000
>> v = inv(G)*i
%v =
%   9.0909
%  10.9091
```

Os resultados obtidos podem ser confirmados por intermédio da Fig. 5.

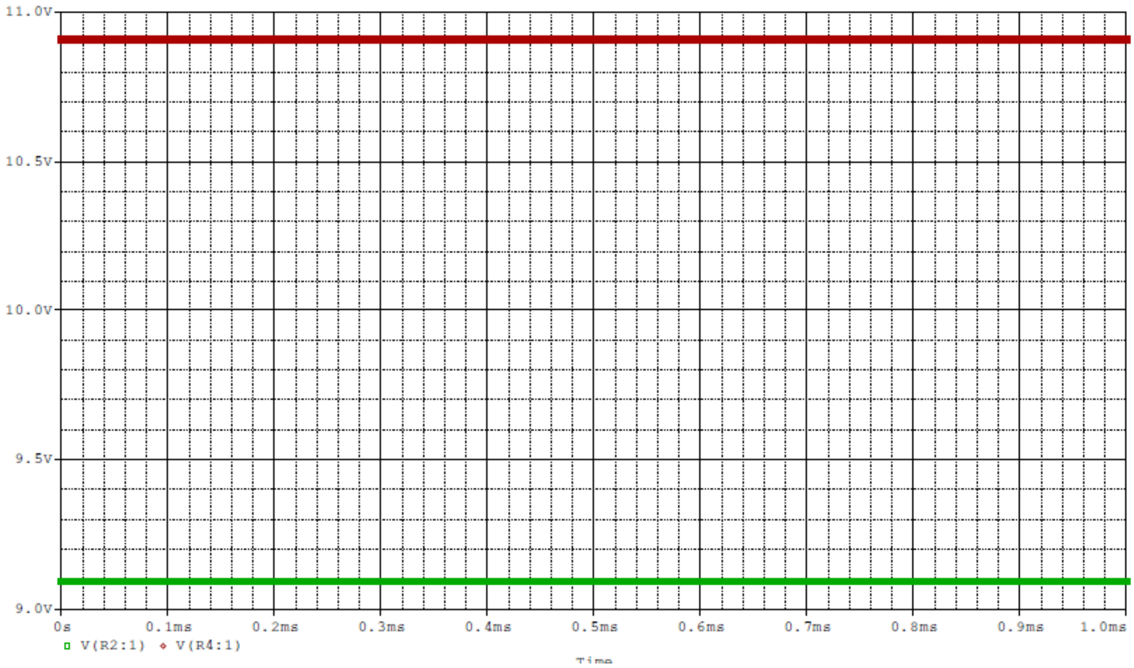


Figura 5 - Formas de onda das tensões nos nós 1 e 2

5. VETORES

5.1. Declaração

É possível trabalhar com vetores no MATLAB, cuja representação é feita baseando-se numa matriz linha. Por exemplo, para obter o vetor (1,3,8), basta iniciarmos com:

```
>> R = [1 3 8]
%G =
%     1      3      8
```

Portanto, todas as operações se tornam possíveis a partir do uso de funções apropriadas. É importante salientar que certas funções exigem a declaração de vetores por matriz coluna, entretanto, nada que uma consulta no *help* para ajudar.

Uma operação básica com vetores é na determinação do número de elementos, a partir da função *length*, assim como no cálculo do seu módulo, usando a função *norm*, ambas definidas abaixo. Logo depois, serão dadas algumas funções que trabalham com vetores.

- ***length***

Definição: Retorna o número de elementos que compõem o vetor.

Sintaxe:

length(A) → Calcula o numero de termos do vetor A

```
>> A = [8 9 5 7];
>> length(A)
%ans =
%     4
```

- ***norm***

Definição: Retorna o módulo do vetor.

Sintaxe:

norm(A) → Calcula o módulo do vetor A.

```
>> x = [0 5 1 7];
>> sqrt(0+5^2+1^2+7^2) %Forma Euclidiana
%ans =
%    8.6603
>> norm(x) %Usando norm
%ans =
%    8.6603
```

5.2. Operações

Quando se deseja calcular o produto vetorial (ou cruzado) de vetores, utiliza-se a função *cross*, apresentada a seguir:

- ***cross***

Definição: Calcula o produto vetorial entre A e B.

Sintaxe:

C = *cross*(A,B) → Retorna, em C, o produto vetorial dos vetores tridimensionais A e B.

De modo análogo, define-se a função dot como a responsável pelo produto escalar de dois vetores dados, conforme definição a seguir.

- ***dot***

Definição: Determina o produto escalar entre dois vetores.

Sintaxe:

C = *dot*(A,B) → Retorna, em C, o produto escalar dos vetores n-dimensionais A e B.

```
>> a = [1 7 3];
>> b = [5 8 6];
>> c = cross(a,b)
%c =
%   18      9    -27
>> d = dot(a,b)
%d =
%   79
```

Além disso, qualquer outra operação é possível, como soma e subtração, mas se deve atentar-se ao fato de que ambos os vetores devem possuir a mesma dimensão.

Dando continuidade, serão definidas algumas funções que poderão ser úteis quando se trabalha com vetores ou até mesmo com matrizes.

- ***min***

Definição: Retorna os valores mínimos de um vetor ou o das colunas de uma matriz.

Sintaxe:

min(A) → Retorna em um vetor linha os menores valores de cada linha da matriz A.

min(A,B) → Retorna uma matriz com os menores valores de cada posição correspondente de ambas as matrizes

[a,b] = *min*(A) → Retorna, em a, os menores valores de cada coluna e, em b, a posição dos mesmos nas suas respectivas colunas.

```
>> A=[0 3; 2 4];
>> [a,b]=min(A)
%a =
%     0     3
%b =
%     1     1
```

- ***max***

Definição: Retorna os valores máximos de um vetor ou o das colunas de uma matriz.

Sintaxe:

max(A) → Retorna em um vetor linha os maiores valores de cada linha da matriz A.

max(A,B) → Retorna uma matriz com os maiores valores de cada posição correspondente de ambas as matrizes.

[a,b] = *max*(A) → Retorna, em a, os maiores valores de cada coluna e, em b, a posição dos mesmos nas suas respectivas colunas.

```
>> A=[0 3; 2 4];
>> [a,b]=min(A)
%a =
%     2     4
%b =
%     2     2
```

- ***sum***

Definição: Calcula o somatório dos elementos de um vetor ou o somatório das colunas de uma matriz.

Sintaxe:

sum(A) → Retorna a soma dos elementos de um vetor ou a soma das colunas de uma matriz.

```
>> A=[0 3; 2 4];
>> sum(A)
%ans =
%     2     7
```

- ***prod***

Definição: Calcula o produtório dos elementos de um vetor ou o produtório das colunas de uma matriz.

Sintaxe:

prod(A) → Retorna o produto dos elementos do vetor A ou o produto das colunas da matriz A.

```
>> A=[0 3; 2 4];
>> prod(A)
%ans =
%    0    12
```

Exercício 5 - Os três vértices de um triângulo estão em A (6,-1,2), B (-2,3,-4) e C (-3,1,5). Determine o vetor unitário perpendicular ao plano no qual o triângulo está localizado. Também determine o ângulo θ_{BAC} no vértice A.

Exercício 6 - Declare a matriz X no MATLAB e determine: $X = \begin{pmatrix} 6 & 2 & 45 \\ 65 & 32 & 9 \\ 3 & -8 & 1 \end{pmatrix}$

Um vetor Y que tenha o valor mínimo das três primeiras colunas;

- **a.** A soma dos elementos de Y;
- **b.** Um vetor Z que tenha o valor máximo das três primeiras linhas;
- **c.** O produtório dos elementos do vetor Z;
- **d.** Calcule o módulo dos elementos de cada linha.

5.3. Sistemas de Coordenadas

Existem funções, no MATLAB, que possibilitam as transformadas de coordenadas, conforme listadas a seguir:

- ***cart2pol***

Definição: Converte do cartesiano para o polar/cilíndrico. Observe a Fig. 6.

Sintaxe:

[theta,rho,z] = *cart2pol*(x,y,z) → Converte o ponto de coordenadas cartesianas (x,y,z) para coordenadas cilíndricas (theta,rho,z).

[theta,rho] = *cart2pol*(x,y) → Converte o ponto de cartesianas (x,y) para coordenadas polares (theta,rho).

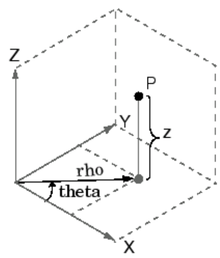


Figura 6 - Transformação entre coordenadas cartesianas e polares/cilíndricas

- ***pol2cart***

Definição: Converte do sistema de coordenadas polares/cilíndricas para o sistema cartesiano.

Sintaxe:

[x,y] = *pol2cart*(theta,rho) → Converte o ponto de coordenadas polares (theta,rho) para coordenadas cartesianas (x,y).

[x,y,z] = *pol2cart*(theta,rho,z) → Converte o ponto de coordenadas cilíndricas (theta,rho,z) para coordenadas cartesianas (x,y,z).

- ***cart2sph***

Definição: Transforma do sistema de coordenadas cartesianas para o sistema de coordenadas esféricas. Observe a Fig. 7.

Sintaxe:

[theta,phi,r] = *cart2sph*(x,y,z) → Converte o ponto de coordenadascartesianas (x,y,z) para coordenadas esféricas (theta,phi,r).

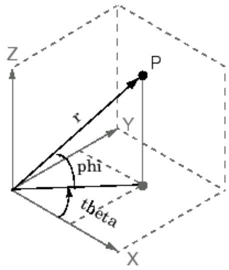


Figura 7 - Transformação entre coordenadas cartesianas e esféricas

- ***sph2cart***

Definição: Transforma do sistema de coordenadas esféricas para o sistema de coordenadas cartesianas.

Sintaxe:

[x,y,z] = *sph2cart*(theta,phi,r) → Converte o ponto de coordenadas esféricas (theta,phi,r) para cartesianas (x,y,z).

Um exemplo para o uso destas funções é na utilização das equações de potenciais elétricos para determinadas distribuições de carga que são simétricos a um sistema de coordenadas. Vejamos o exemplo abaixo.

Exemplo 2 - Um dipolo elétrico é formado colocando uma carga de 1 nC em (1,0,0) e uma outra carga de -1 nC em (-1,0,0). Determine as linhas equipotenciais geradas a partir dessa configuração.

```
>> [x,y,z] = meshgrid(-0.5:.012:0.5);
>> [teta,fi,r] = cart2sph(x,y,z);
>> v = (1e-9*0.2*cos(teta))./(4.*pi.*8.85e-12.*r.^2);
>> contourslice(x,y,z,v,[ -0.5:0.5],[-0.5:0.5],[-0.5:0.5]);
>> colormap hsv
>> grid on
```

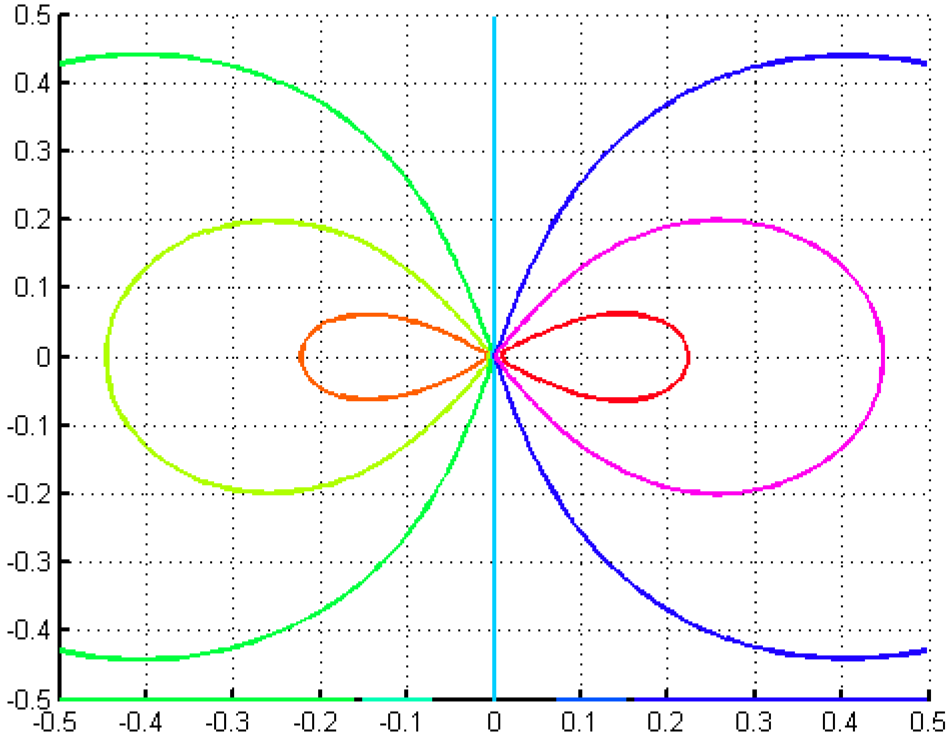


Figura 8 - Linhas equipotenciais

6. M-FILE

6.1. Definição

O M-File é uma ferramenta do MATLAB que auxilia na criação de funções e *scripts*. Para criar um novo M-File deve-se entrar na aba *Home* no menu *New* e escolher a opção *script*, Fig. 9, ou clicar em *New Script* na aba *Home*:

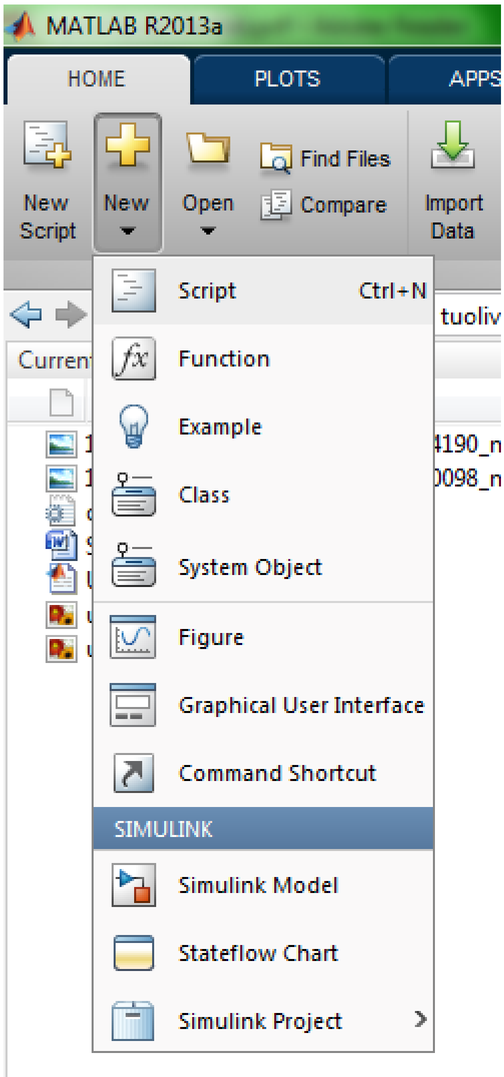


Figura 9 - Criando um novo script pelo menu *new*

Será aberto um editor de textos. Este é mais um ambiente de programação do MATLAB. Os comandos usados nesse editor são os mesmos do *Command Window* e podem ser utilizadas as variáveis já presentes no *Workspace*. Uma grande utilidade do M-File é facilitar a vida do usuário, através da execução de *scripts*, que são arquivos que possuem uma lista de comandos que devem ser executados seqüencialmente (também chamados de procedimentos). Assim, quando se quiser executar mais de uma vez um conjunto de comandos seqüenciais relativamente grandes e trabalhosos para serem digitados, não

será necessário digitá-los um a um na *Command Window*, basta digitar os comandos no M-File, salvar o arquivo e executá-lo. Segue um exemplo de *script* para calcular a distância entre dois pontos:

```
p=[1 2];
q=[4 5];
temp=((p(1)-q(1))^2+(p(2)-q(2))^2);
distancia=sqrt(temp);
```

O primeiro passo para executar um *script* é salvá-lo. O M-File salva os arquivos do M-File no seu diretório padrão, com a extensão .m. Para salvar deve-se entrar no menu *File* e escolher a opção *Save*. Existem duas opções para se executar o *script*, a primeira é clicar no botão *Run*, que está localizado na Barra de Ferramentas *Run* do M-File, Fig. 10, em verde:

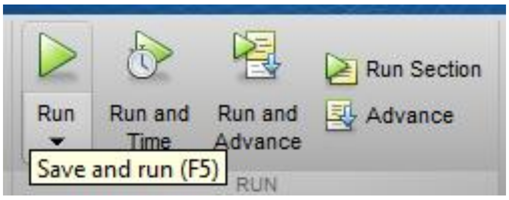


Figura 10 - Botão *Run*

A outra opção é usar o atalho F5. As conseqüências da execução são as mesmas que seriam observadas se os comandos tivessem sido executados na *Command Window*: as variáveis declaradas estarão todas disponíveis no *Workspace* e os resultados de cada operação estarão presentes na tela.

O M-FILE é um ambiente de programação, portanto existem algumas funções, bem parecidas com as da linguagem C, por exemplo, que podem ser utilizadas.

6.2. Organização

Para uma melhor organização podemos fazer comentários utilizando o símbolo '%', ou selecionando o texto inteiro e teclando Crtl+R, ou '%{' para abrir o comentário por bloco e '%}' para fechar.

Podemos ainda utilizar o símbolo '%%' para que, no mesmo M-File, o usuário possa rodar apenas algumas partes do programa, Fig. 11. Para rodar somente a parte selecionada, tecle Ctrl+Enter e para rodar o programa inteiro clique em F5 ou no ícone da Fig. 12.

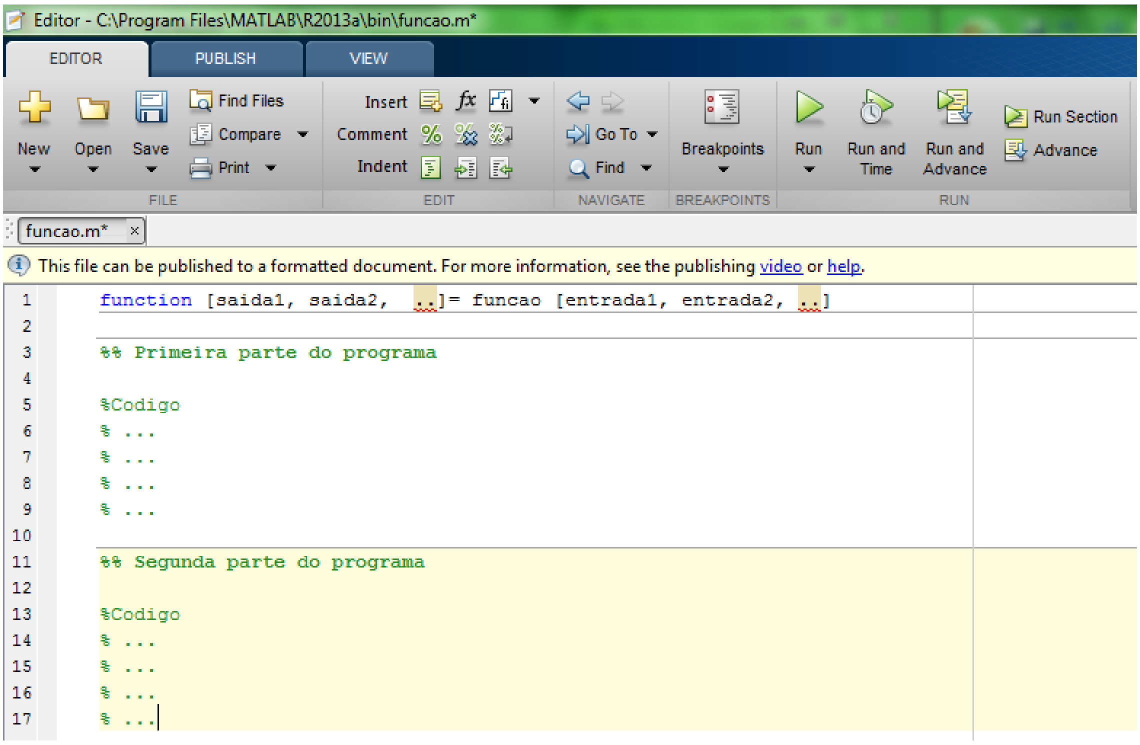


Figura 11 - *Script* com comentários



Figura 12 - Iniciar simulação

6.3. Operações no M-File

- ***if***

Definição: Operação condicional. Executa as funções contidas no comando. Pode ser utilizado com *else*, que executa caso a condição declarada for falsa, e com *elseif*, que executa a função caso outra condição posteriormente declarada for verdadeira.

Para mais de uma condição, utiliza-se para "e", &&, e para "ou", ||.

Sintaxe:

if <condição>

[Comandos]

elseif <condição>

[Comandos]

else

[Comandos]

end

- ***for***

Definição: Comando de iteração. Permite que um conjunto de instruções seja executado até que a condição seja satisfeita.

Sintaxe:

for <condição>

[Comandos]

end

- ***while***

Definição: Comando de iteração. Executa um bloco de instruções enquanto a condição for verdadeira.

Sintaxe:

while <condição>

[Comandos]

end

- ***switch***

Definição: Operação condicional. Testa sucessivamente o valor da expressão dada e direciona para o caso especificado. Funciona como um bloco de "*if's*"

Sintaxe:

switch <condição>

case caso1

[Comandos]

case {caso1, caso2, caso3, ...}

[Comandos]

otherwise (Caso não seja nenhuma das outras condições)

[Comandos]

end

```
>> a = 3;
>> switch a
>> case {2}
>> disp('Resposta um')
>> case {3}
>> disp('Resposta dois')
>> case '5'
>> disp('Resposta tres')
>> otherwise
>> disp('Resposta ?')
>> end
```

- ***disp***

Definição: "Escreve" no command window um texto ou o valor de um vetor, sem escrever seu nome.

Sintaxe:

disp(x)

- ***input***

Definição: Pede uma entrada do usuário pelo command window.

Sintaxe:

entrada = *input*('O que deseja?')

```
>> X = input('Entre um número\n')
>> num = 10*X
```

Command Window:

```
Entre um número
23
X =
    23
num =
   230
```

- ***break***

Definição: Quebra um laço for ou while.

Sintaxe:

break

```
>> for i = 0:5
>>     i
>>     if i == 1
>>         break
>>     end
>> end
```

Command Window:

```
i =
     0
i =
     1
```

- ***continue***

Definição: Passa para o próximo laço de um *for* ou *while*.

Sintaxe:

continue

```
>> for i = 0:3
>>     if i == 1 || i == 2
>>         continue
>>     end
>>     i
>> end
```

Command Window:

```
i =
     0
i =
     3
```

- **Operadores Lógicos**

Definição: Operadores lógicos

Tabela 2 - Operadores Lógicos

Entradas		*and*	*or*	*not*	*xor*
A	B	A & B	A | B	~A	xor(A,B)
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

- **Funções em M-File**

Outra importante função do M-File é a criação de funções. A declaração inicial é da seguinte forma:

```
function [saida1, saida2, ...] = nome(entrada1, entrada2, ...)
%declaração do código
...
```

Segue um exemplo:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Função exemplo
% A função recebe um vetor qualquer e retorna dois valores
% vetor2 = vetor multiplicado por 2
% e v1 = o valor do primeiro elemento do vetor
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [vetor2, v1]= funcao(vetor)
vetor2=vetor*2; % multiplica o vetor por 2
v1=vetor(1); % retorna o primeiro elemento do vetor de entrada...
```

Para chamar a função, basta digitar na janela de comando o nome da função com as entradas entre parênteses. <u>É importante lembrar de salvar o arquivo com mesmo nome da função!</u>

Na janela de comandos do MATLAB podemos colocar um vetor como exemplo:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
>> A = [2 5 -8 4 1 6]
%A =
%      2      5     -8      4      1      6
>> [x,y]=funcao(A)
%x =
%     4     10    -16      8      2     12
%y =
%     2
function [vetor2, v1]= funcao(vetor)
vetor2=vetor*2; % multiplica o vetor por 2
v1=vetor(1); % retorna o primeiro elemento do vetor de entrada...
```

7. FUNÇÕES MATEMÁTICAS

7.1. Funções Elementares

O MATLAB contém um conjunto de funções que executam algumas funções matemáticas elementares, como módulo e raiz quadrada. A seguir disponibilizaremos uma lista de funções com uma breve descrição, Tabela 3.

Tabela 3 - Funções matemáticas elementares

Função	Descrição
log(X)	Determina o logarítmo natural de X
log10(X)	Determina o logarítmo de X na base 10
log2(X)	Calcula o logarítmo de X na base 2
exp(X)	Determina a exopressão de ex
sqrt(X)	Retorna a raiz quadrada de X

7.2. Propriedades Fundamentais

O MATLAB possui também funções que possibilitam os cálculos elementares de matemática, tais como mmc, mdc e entre outros. Por exemplo, as funções *gcd* e *lcm* retornam o máximo divisor comum e o mínimo múltiplo comum, respectivamente. Vejamos abaixo essas e outras funções similares:

- ***gcd***

Definição: Determina o máximo divisor comum entre dois parâmetros.

Sintaxe:

G = *gcd*(A,B) → Retorna, em G, o máximo divisor comum entre A e B.

- ***lcm***

Definição: Determina o mínimo múltiplo comum entre dois parâmetros.

Sintaxe:

G = *lcm*(A,B) → Retorna, em G, o mínimo múltiplo comum entre A e B.

- ***factorial***

Definição: Retorna o fatorial de um argumento.

Sintaxe:

factorial(N) → Calcula o fatorial de N.

- ***nchoosek***

Definição: Desenvolve a fatoração binomial.

Sintaxe:

C = *nchoosek*(n,k) → Retorna o número de combinações de n tomada k a k.

- ***primes***

Definição: Devolve uma lista com uma quantidade desejada de números primos.

Sintaxe:

p = *primes*(n) → Devolve uma lista com n de números primos.

- ***mod***

Definição: Calcula a congruência entre dois argumentos.

Sintaxe:

M = *mod*(X,Y) → Retorna, em M, a congruência entre os argumentos X e Y.

- ***rem***

Definição: Determina o resto da divisão de dois argumentos.

Sintaxe:

R = *rem*(X,Y) → Retorna, em R, o resto da divisão de X por Y.

- ***perms***

Definição: Desenvolve todas as permutações possíveis dos argumentos dados.

Sintaxe:

P = *perms*(v) → v pode ser uma matriz com os números nos quais deseja permutá-los.

7.3. Números Complexos

O MATLAB proporciona um conjunto de funções que auxilia o manuseio de números complexos. Inicialmente, para definir um número complexo utilizam-se os operadores i e j (voltado mais para a engenharia). Por exemplo, para definir a = 5 + 8i , faz-se:

```
>> a = 5 +8i
%a =
%    5.0000 + 8.0000i
>> a = 5 + 8j
%a =
%    5.0000 + 8.0000i
```

Há outra forma de definir um número complexo no MATLAB, através da função *complex*. A sua vantagem está na maior liberdade que se tem para alterar a parte imaginária, real, módulo ou até mesmo a fase do número, no ponto de vista computacional. A definição dessa função é descrita como:

- ***complex***

Definição: Retorna um número complexo a partir da sua parte real e da sua parte imaginária.

Sintaxe:

c = *complex*(a,b) → Retorna, em c, o número complexo de parte real a e parte imaginária b.

Quando se deseja trabalhar com módulo, ângulo de fase, conjugado, ou entre outros, tornam-se fáceis de serem calculados quando se utiliza a função adequada. Na Tabela 4 serão denotadas algumas funções que possibilitam isso.

Tabela 4 - Funções para números complexos

Função	Descrição
abs(X)	Retorna o módulo do número complexo x
angle(X)	Retorna a fase do comlexo X
conj(X)	Calcula o conjugado do número complexo X
imag(X)	Determina a parte imaginária de X
real(X)	Determina a parte real de X

Exercício 7 - Determine todos os parâmetros intrínsecos ao número complexo $9e^{(5+3i)}$

7.4. Funções Trigonométricas

Quando trabalhamos com trigonometria, o MATLAB dispõe de funções que operam neste ramo matemático. Tabela 5 resume bem algumas funções que possuem este fim.

Tabela 5 - Funções trigonométricas.

Função	Descrição
cos(X)	Cosseno do argumento X em radianos
sin(X)	Seno do argumento X em radianos
tan(X)	Tangente do argumento X em radianos
sec(X)	Secante do argumento X em radianos
csc(X)	Cossecante do argumento X em radianos
cot(X)	Cotangente do argumento X em radianos

Veja acima que estas funções retornam um valor correspondente a um argumento em radianos. Quando for desejado entrar com um argumento em grau, basta utilizar o sufixo d em cada função. Por exemplo, o seno de 30°:

```
>> sind(30)
%ans =
%     0.5000
```

Entretanto, quando deseja calcular o arco correspondente a um valor pra uma dada função, basta utilizar o prefixo *a* diante as funções. Como exemplo, determinemos o arco-tangente de 1 em radianos:

```
>> atan(1)
%ans =
%     0.7854
```

Caso queiramos saber em grau, faríamos:

```
>> atand(1)
%ans =
%   45
```

Por fim, quando se deseja determinar a função hiperbólica, basta utilizar o sufixo h na função dada. Vejamos no comando a seguir:

```
>> cosh(3)
%ans =
%   10.0677
```

De fato, o resultado é coerente, pois:

```
>> (exp(3)+exp(
%ans =
%   10.0677
```

A Tabela 6 resume bem o uso de sufixo e prefixo em funções trigonométricas:

Tabela 6 - Sufixos e prefixos de funçõs trigonométricas

Prefixo	Sufixo	Descrição	Exemplo
a	-	Determina o arco de um valor	*atan*
-	*d*	Determina o argumento em graus	*cosd*
-	*h*	Determina a função hiperbólica	*sinh*

7.5. Aproximação Inteira

Na biblioteca de funções do MATLAB, há uma variedade que trabalha no intuito do arredondamento de números. Indubitavelmente, a mais importante é a *round*, que arredonda para o inteiro mais próximo. Obviamente, esta importância depende do ambiente prático no qual a função está sendo submetida. Abaixo segue uma lista de funções que tratam com aproximações numéricas.

- ***round***

Definição: Arredonda os elementos de uma matriz ou de um vetor para o inteiro mais próximo desses elementos. Também é válido para números complexos.

Sintaxe:

Y = *round*(X) → Retorna, em Y, os inteiros mais próximos dos elementos de X.

- ***ceil***

Definição: Arredonda os elementos de uma matriz ou de um vetor para o inteiro imediatamente maior que os respectivos elementos.

Sintaxe:

B = *ceil*(A) → Retorna, em B, os inteiros imediatamente maiores que os elementos de A.

- ***floor***

Definição: Arredonda os elementos de uma matriz ou de um vetor para o inteiro imediatamente menor que os respectivos elementos.

Sintaxe:

B = *floor*(A) → Retorna, em B, os inteiros imediatamente menores que os elementos de A.

- ***fix***

Definição: Arredonda os elementos de uma matriz ou de um vetor para o inteiro mais próximo de tal modo que esteja em direção ao zero.

Sintaxe:

B = *fix*(A) → Retorna, em B, os inteiros mais próximos, em direção ao zero, dos elementos de A.

8. GRÁFICOS

8.1. Gráficos Bidimensionais

- ***ezplot***

Definição: Plota a expressão simbólica f(x) no domínio padrão $-2\pi<x<2\pi$. Observe a Fig. 13.

Sintaxe:

ezplot(t) → Plota a expressão f(x).

```
>> ezplot('sin(x)')
```

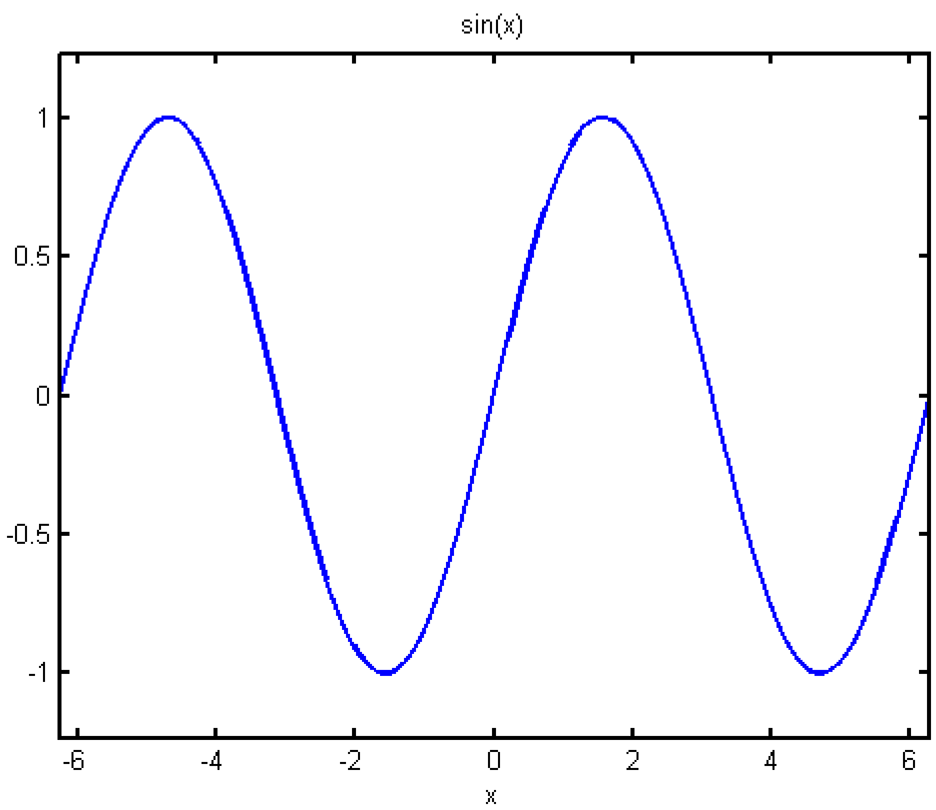


Figura 13 - Gráfico *sin(x)* gerado pela função *ezplot*.

- ***plot***

Definição: Plota as colunas de um vetor versus os índices de cada elemento, se o vetor for real. Se for complexo, plota a parte real pela parte imaginária de cada elemento. Observe a Fig. 14.

Sintaxe:

plot(X) → Se X for real, plota as colunas de X pelos índices de cada elemento.

plot(X) → Se X for complexo, plota a parte real pela parte imaginária de cada elemento. É equivalente a *plot(real(X),imag(X))*.

plot(X,Y) → Plota os elementos de X pelos de Y.

```
>> t = 0:pi/50:10*pi;
>> plot(t,sin(t
```

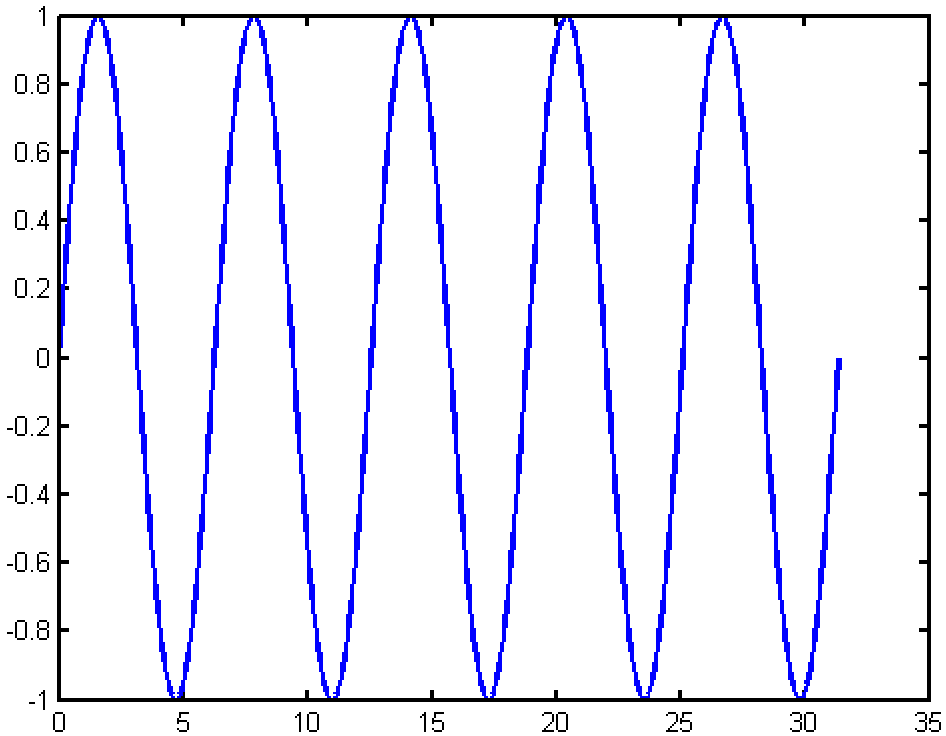


Figura 14 - Gráfico *sin(t)* gerado pela função *plot*.

- ***line***

Definição: Cria uma linha no gráfico atual. Observe a Fig. 15.

Sintaxe:

line(X,Y) → Cria uma linha definida nos vetores X e Y no gráfico atual.

line(X,Y,Z) → Cria uma linha no espaço tridimensional.

```
>> x= -2:0.01:5;
>> line(x,exp(x))
```

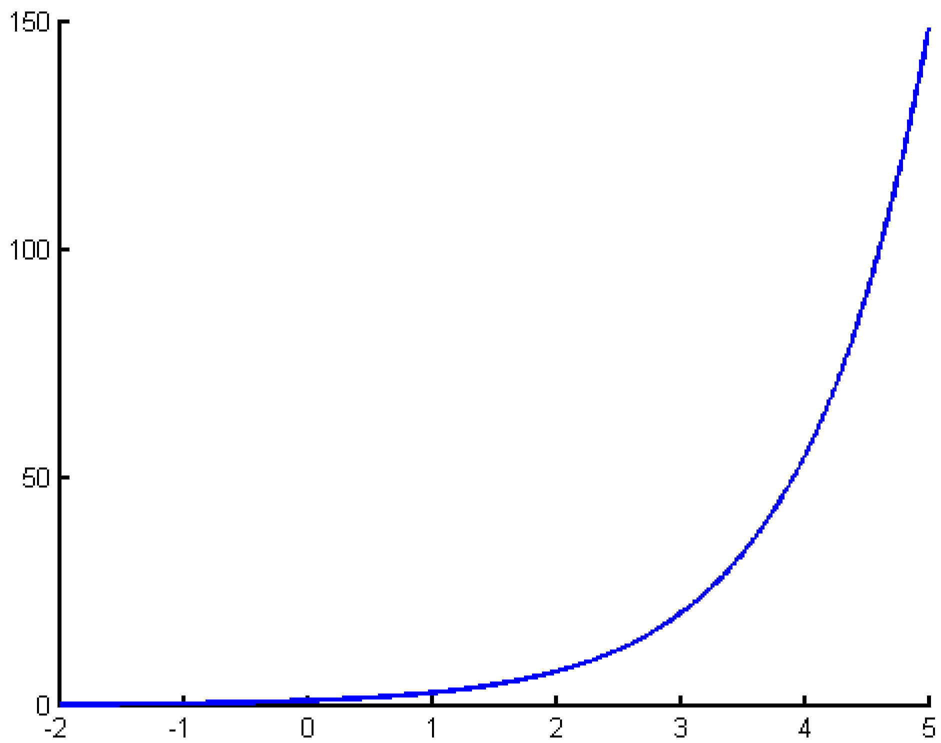


Figura 15 - Gráfico e^x gerado pela função *line*.

- ***stem***

Definição: Plota uma seqüência de dados discretos. Observe a Fig. 16.

Sintaxe:

stem(Y) → Plota a seqüência de dados do vetor Y em um domínio discreto ao longo do eixo-x.

stem(X,Y) → Plota X em função de Y em um domínio discreto. X e Y devem ser vetores ou matrizes de mesmo tamanho.

```
>> x = -4:4;
>> y = exp(x);
>> stem(y(
```

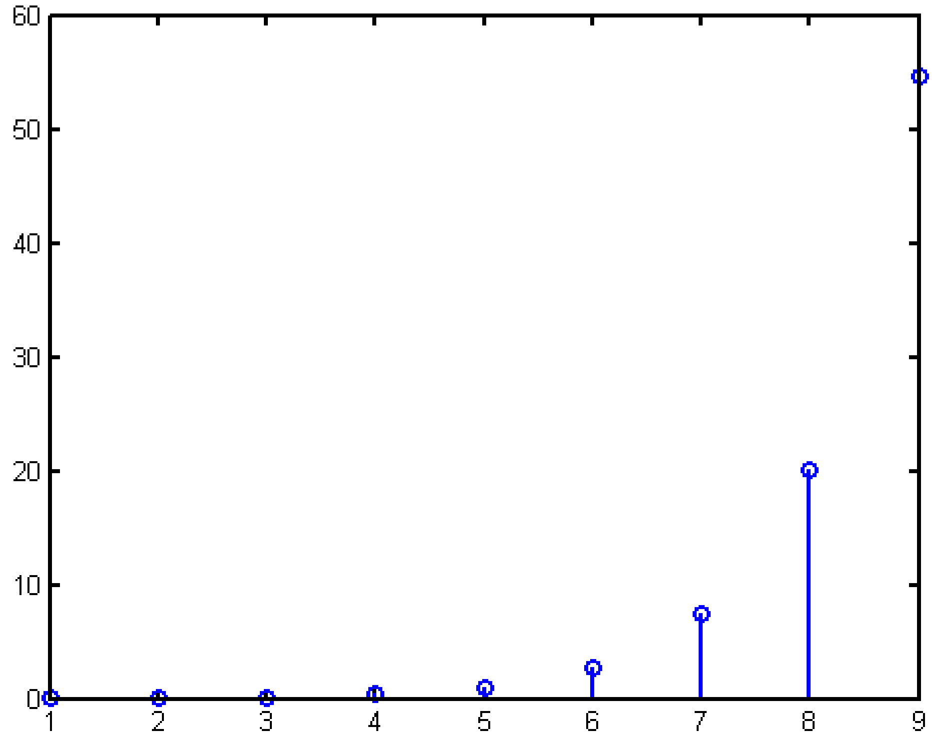


Figura 16 - Gráfico e^x gerado pela função *stem*.

- ***compass***

Definição: Plota vetores de componentes cartesianas a partir da origem de um gráfico polar. Observe a Fig. 17.

Sintaxe:

compass(U,V) → Plota o vetor de componentes cartesianas U e V partindo da origem do gráfico polar.

```
>> compass(2,3)
```

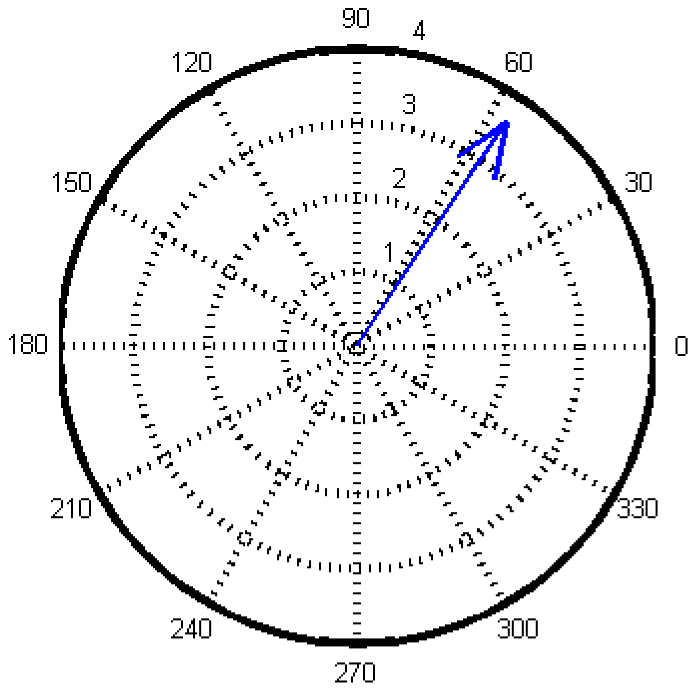


Figura 17 - Gráfico polar gerado pela função compass.

8.2. Gráficos Tridimensionais

- ***ezplot3***

Definição: Plota uma curva espacial de três equações paramétricas no domínio padrão $0 < t < 2\pi$. Observe a Fig. 18.

Sintaxe:

ezplot3(x,y,z) → Plota a curva paramétrica x = x(t) , y = y(t) e z = z(t) .

```
>> ezplot3('cos(t)','sin(t)','t')
```

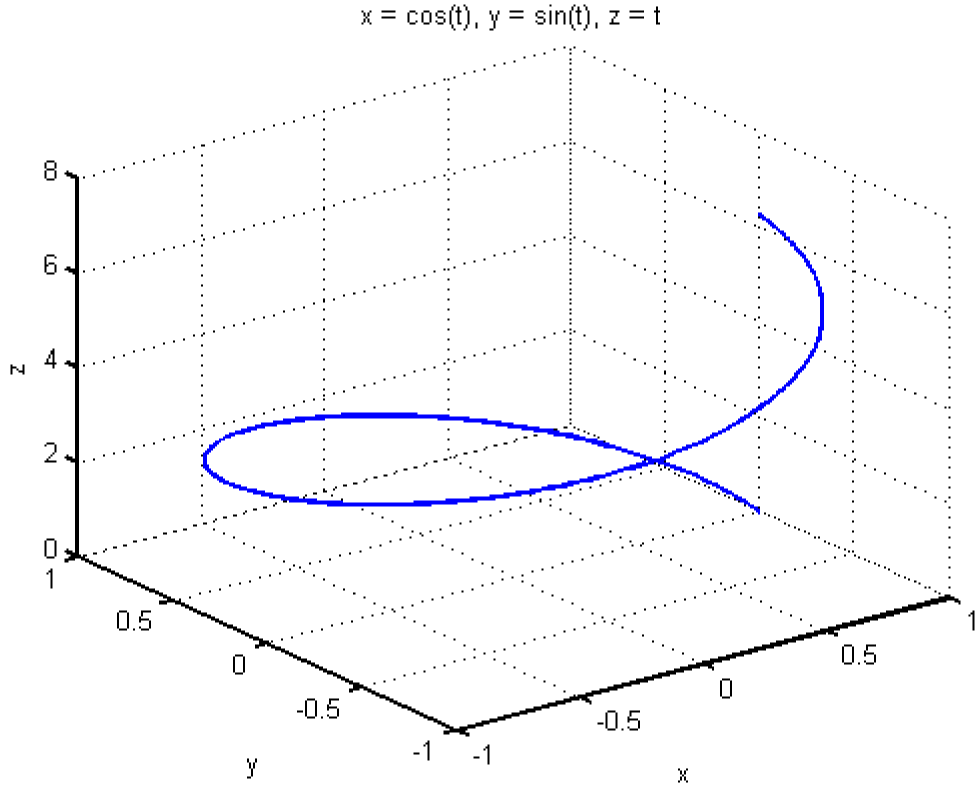


Figura 18 - Gráfico *cos(t)*, *sin(t)*, t gerado pela função *ezplot3*.

- ***plot3***

Definição: Plota tridimensionalmente um gráfico. Observe a Fig. 19.

Sintaxe:

Plot3(X,Y,Z) → Plota uma ou mais linhas no espaço tridimensional através de pontos cujas coordenadas são elementos dos vetores ou matrizes X,Y e Z.

```
>> >> t = 0:pi/50:10*pi;
>> plot3(cos(t),sin(t),t)
```

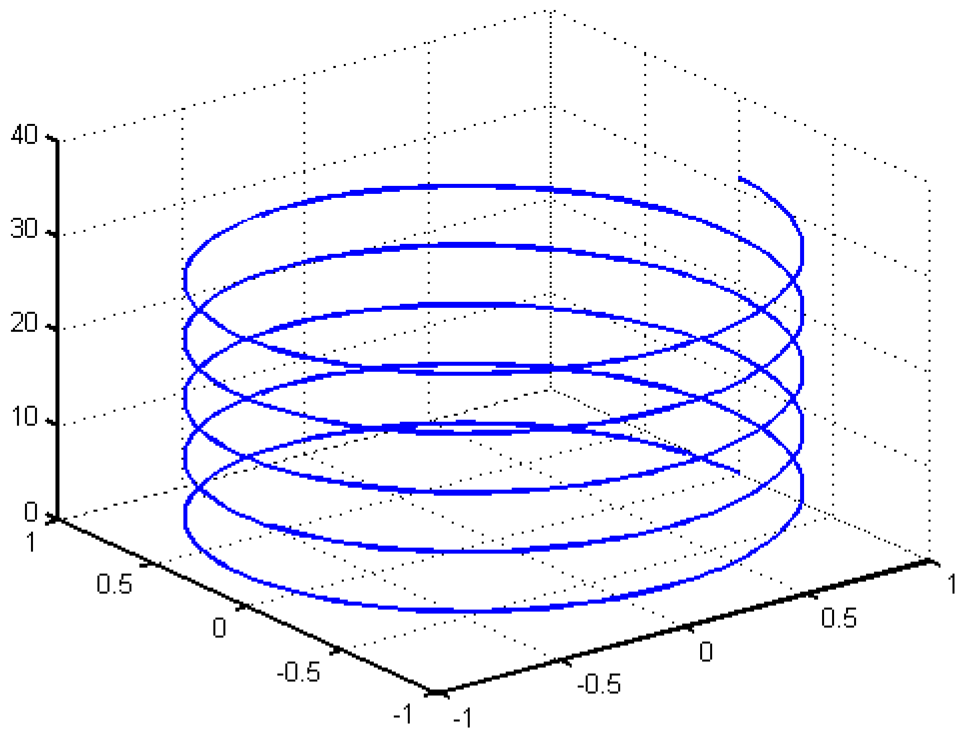


Figura 19 - Gráfico *cos(t)*, *sin(t)*, t gerado pela função *plot3*.

- ***ezsurf***

Definição: Plota a superfície de um gráfico de uma função de duas variáveis no domínio padrão -2π < x < 2π e -2π < y < 2π. Observe a Fig. 20.

Sintaxe:

ezsurf(X,Y,Z) → Plota a superfície paramétrica x = x (s, t) , y = y (s, t) e z = z (s, t) no domínio -2π < s < 2π e -2π < t < 2π

```
>> ezsurf('1/sqrt(x^2 + y^2)')
```

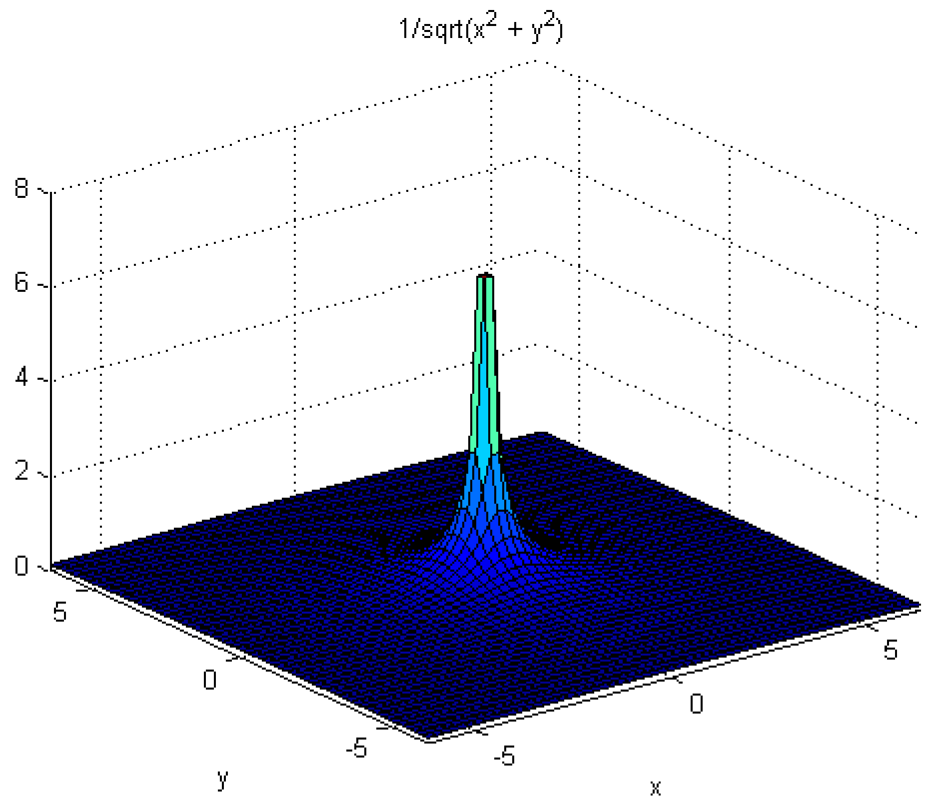


Figura 20 - Superfície $\frac{1}{\sqrt{x^2+u^2}}$ gerada pela função *ezsurf*.

- ***meshgrid***

Definição: Prepara a criação de uma superfície de um gráfico tridimensional.

Sintaxe:

[X,Y] = *meshgrid(x,y)* → Transforma o domínio especificado pelos vetores x e y em matrizes de vetores X e Y, as quais podem ser usadas para preparar a plotagem de superfície de um gráfico tridimensional de uma função de duas variáveis.

```
>> [X,Y] = meshgrid(-6:0.1:6,-6:0.1:6);
>> Z = 1./(sqrt(X.^2+Y.^2));
```

- ***surf***

Definição: Plota a superfície de um gráfico de uma função de duas variáveis cujo domínio é determinado pelo usuário. Observe a Fig. 21.

Sintaxe:

surf(X,Y,Z) → Plota a superfície paramétrica de Z em função de X e Y.

```
>> surf(X,Y,Z)
```

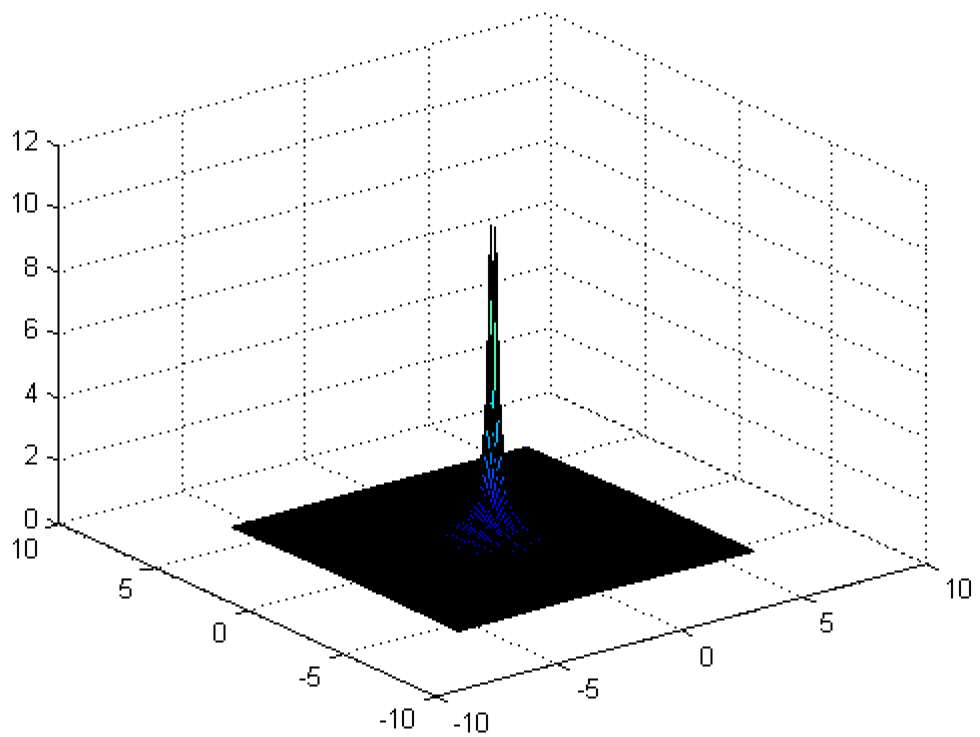


Figura 21 - Superfície $\frac{1}{\sqrt{x^2+u^2}}$ gerada pela função *surf*.

8.3. Configuração

- ***text***

Definição: Cria objetos de texto em locais específicos do gráfico. Observe a Fig. 22.

Sintaxe:

text(x,y,'string') → *Escreve string* no local (x,y). Pode-se modificar *string* das mais diversas formas.

```
>> plot(0:pi/20:2*pi,sin(0:pi/20:2*pi))
>> text(pi,0,'\leftarrow sin(\pi)','FontSize',18)
```

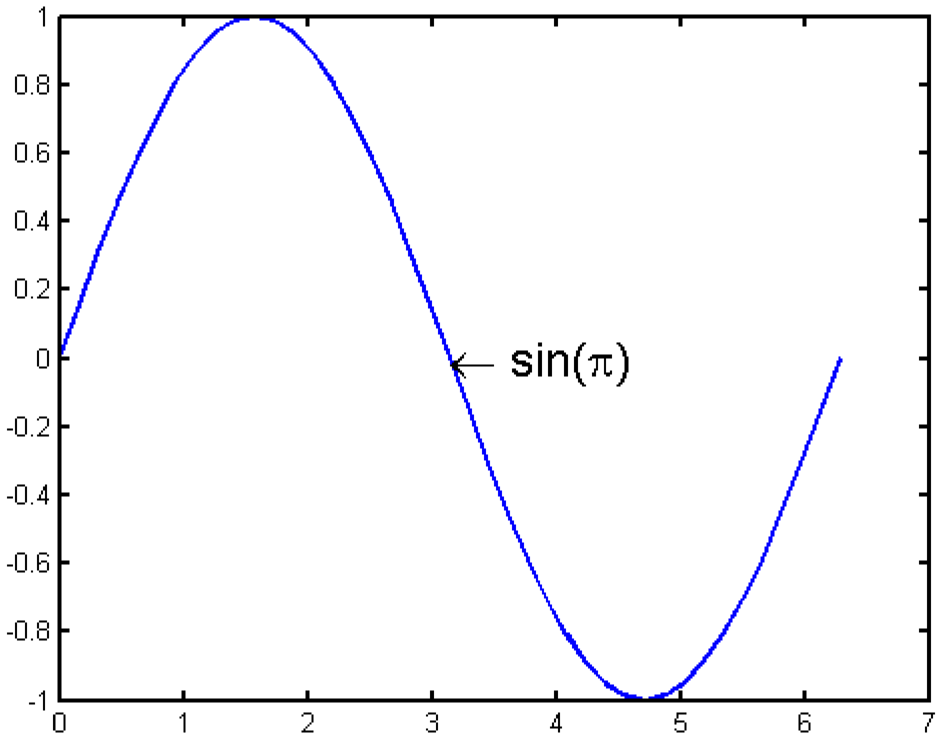


Figura 22 - Gráfico *sin(x)* gerado pela função *plot* com texto gerado pela função *text*.

- ***title***

Definição: Dá um título ao gráfico. Observe a Fig. 23.

Sintaxe:

title('string') → Dá o título *string* ao gráfico atual.

```
>> compass(2,3)
>> title('Gráfico Polar')
```

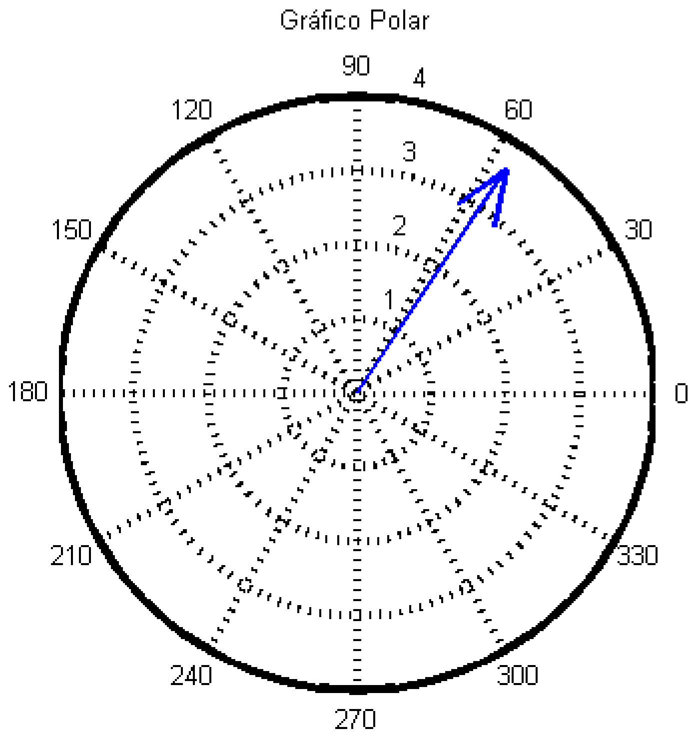


Figura 23 - Gráfico polar gerado pela função *compass* com título gerado pela função

- ***axis***

Definição: Determina os limites dos eixos coordenados X, Y e Z. Observe a Fig. 24.

Sintaxe:

axis([*xmin xmax ymin ymax zmin zmax*]) → define o eixo X de *xmin* a *xmax*, o eixo Y de *ymin* a yimax e o eixo Z de *zmin* a *zmax*.

```
>> t = 0:pi/50:10*pi;
>> plot3(cos(t),sin(t),t)
>> axis([-1.5 1.5 -1.5 1.5 -1 34])
```

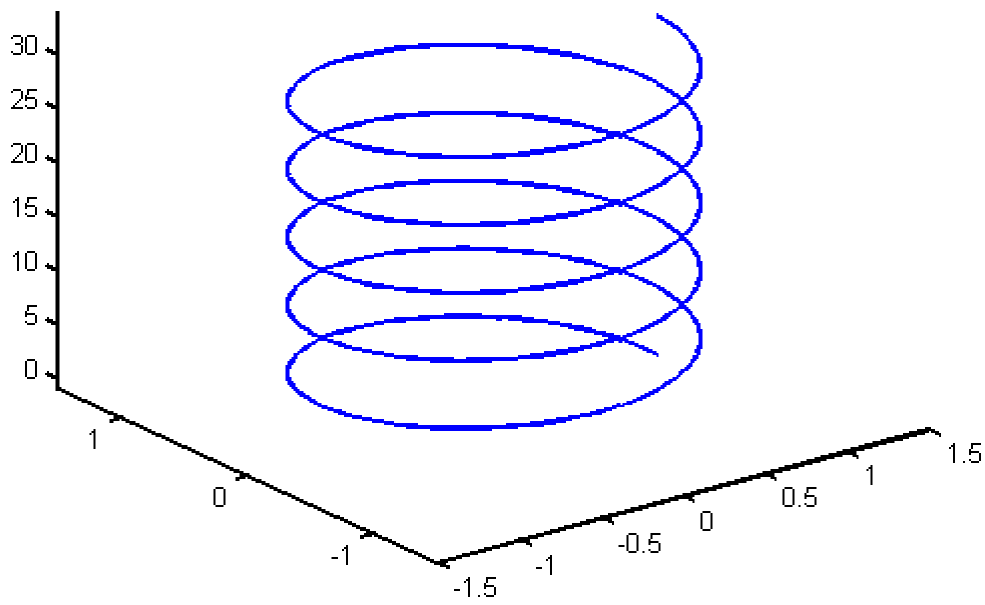


Figura 24 - Gráfico *cos(t)*, *sin(t)*, t gerado pela função *plot3* com eixos ajustados pela função *axis*.

- ***grid***

Definição: Adiciona ou remove as linhas de grade em um gráfico. Observe a Fig. 25.

Sintaxe:

grid on → Adiciona as linhas de grade em um gráfico.

grid off → Remove as linhas de grade em um gráfico.

```
>> t = 0:pi/50:10*pi;
>> plot3(cos(t),sin(t),t)
>> grid on
```

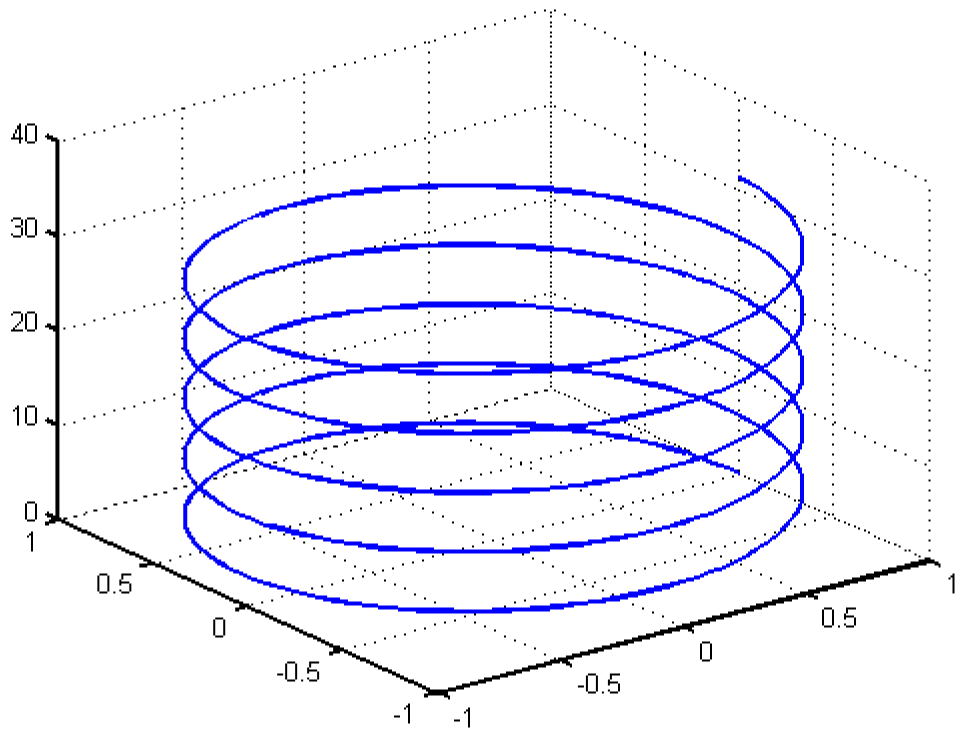


Figura 25 - Gráfico cos(t), sin(t), t gerado pela função plot3 com linhas de grade geradas pela função grid.

- ***hold***

Definição: Determina se objetos são adicionados ao gráfico ou se substituem o existente. Observe a Fig. 26.

Sintaxe:

hold on → Adiciona objetos no mesmo gráfico

hold off → Substitui os objetos existentes em um gráfico pelos atuais.

```
>> x = -6:0.01:6;
>> y = sin(x);
>> plot(x,y)
>> hold on
>> t = -6:0.01:2;
>> k = exp(t);
>> plot(t,k)
```

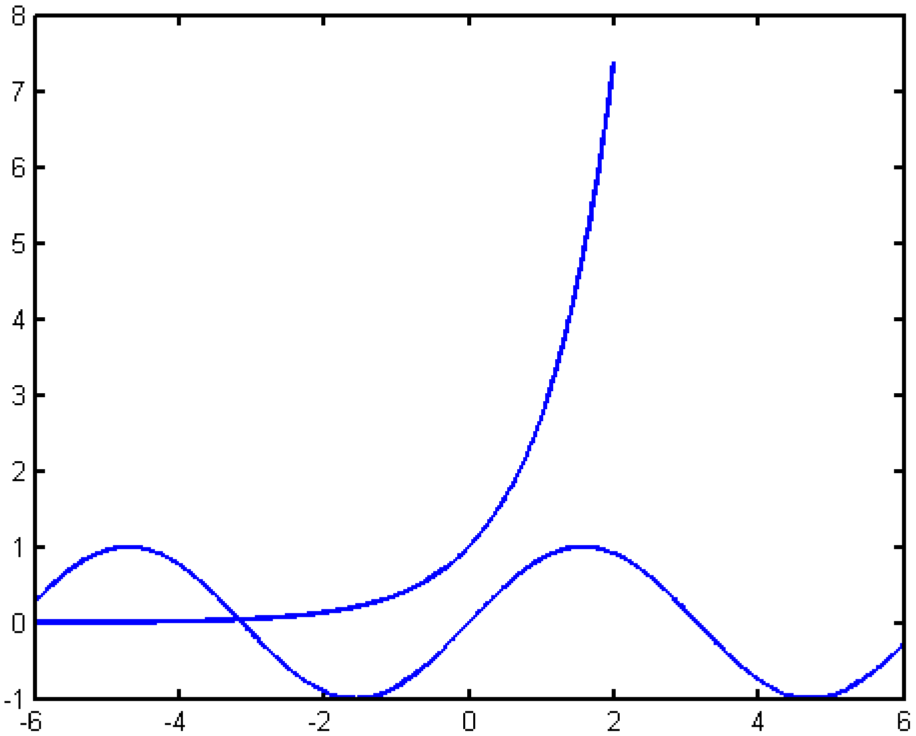


Figura 26 - Gráficos *sin(x)* e e^x gerados pela função *plot* e *ezplot* respectivamente e colocados na mesma janela de gráfico pela função *hold*.

- ***legend***

Definição: Adiciona uma legenda ao gráfico. Observe a Fig. 27.

Sintaxe:

legend('string1','string2') → Adiciona as legendas *string1* e *string2* ao gráfico atual.

```
>> x=-6:0.01:6;
>> y=sin(x);
>> plot(x,y)
>> hold on
>> t=-6:0.01:2;
>> k=exp(t);
>> plot(t,k)
>> legend('Gráfico 1: y=sen(x)','Gráfico 2: y=exp(x)')
```

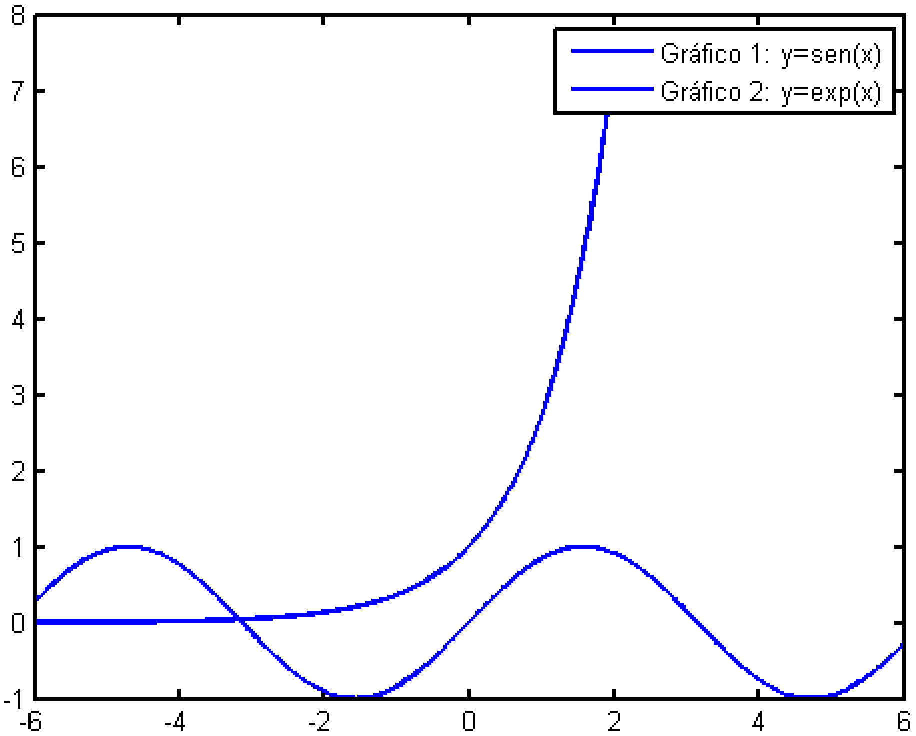


Figura 27 - Gráficos *sin(x)* e e^x gerados pela função *plot* com legenda.

- ***xlabel, ylabel, zlabel***

Definição: Dá um título aos eixos X, Y e Z. Observe a Fig. 28.

Sintaxe:

xlabel('string') → Dá o título *string* ao eixo X.

ylabel('string') → Dá o título *string* ao eixo Y.

zlabel('string') → Dá o título *string* ao eixo Z.

```
>> t = 0:pi/50:10*pi;
>> plot3(cos(t),sin(t),t)
>> xlabel('x=cos(t)')
>> ylabel('y=sin(t)')
>> zlabel('z=t')
```

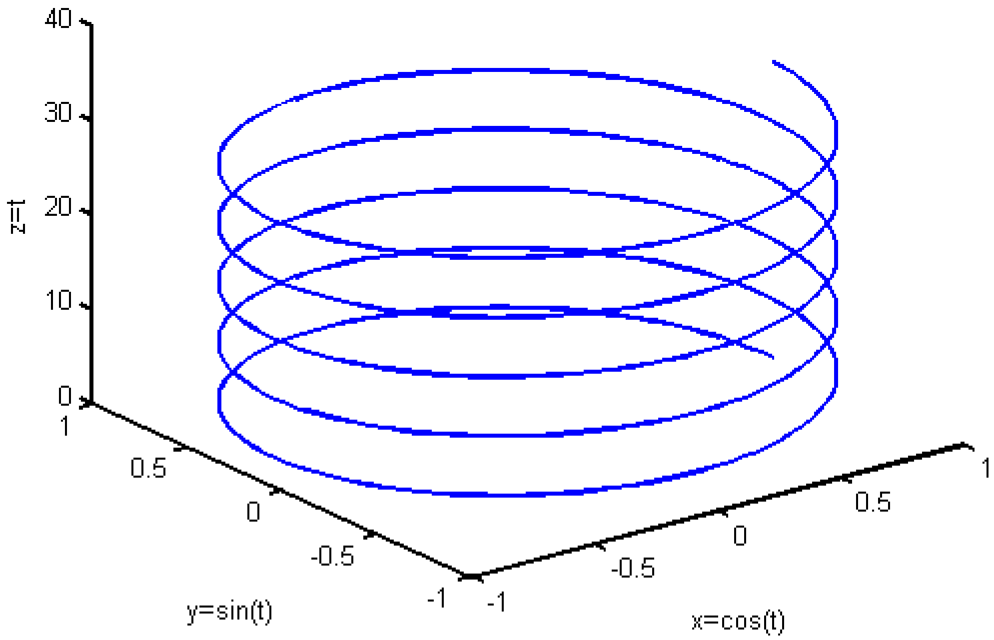


Figura 28 - Gráfico *cos(t)*, *sin(t)*, t gerado pela função *plot3* com títulos nos eixos coordenados.

- ***xlim, ylim, zlim***

Definição: Estipula os limites dos eixos X,Y e Z. Observe a Fig. 29.

Sintaxe:

xlim([xmin xmax]) → define o eixo X de *xmin* a *xmax*.

ylim([ymin ymax]) → define o eixo Y de *ymin* a *ymax*.

zlim([zmin zmax]) → define o eixo Z de *zmin* a *zmax*.

```
>> t = 0:pi/50:10*pi;
>> plot3(cos(t),sin(t),t)
>> xlim([-1.5 1.5])
>> ylim([-1.5 1.5])
>> zlim([-1 34])
```

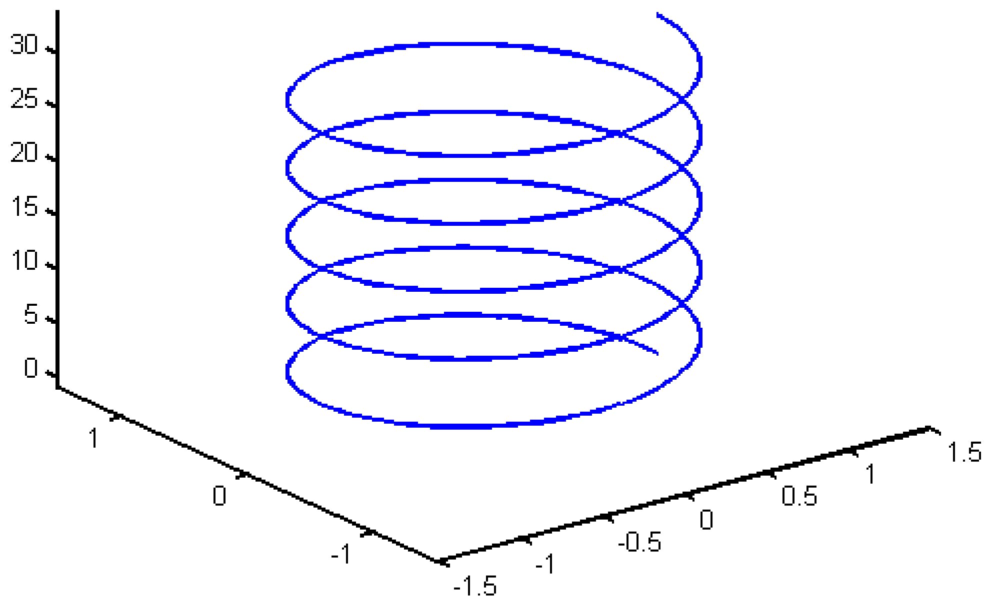


Figura 29 - Gráfico *cos(t), sin(t)*, t gerado pela função *plot3* com eixos ajustados pelas funções *xlim*, *ylim* e *zlim*.

- ***figure***

Definição: Cria uma nova janela para plotar gráficos.

Sintaxe:

figure → Abre uma nova janela de gráfico, definindo-a como janela atual.

- ***subplot***

Definição: Divide a janela do gráfico em uma matriz definida pelo usuário, podendo trabalhar com qualquer um. Observe a Fig. 30.

Sintaxe:

h = *subplot(m,n,p)* (ou *subplot(mnp)*) → Divide em m linhas, n colunas, plotando o gráfico na posição p. Caso tenha uma matriz retangular, a contagem inicia-se no sentido anti-horário do gráfico superior esquerdo.

subplot(m,n,p,'replace') → Se o gráfico já existe, deleta o gráfico especificado, substituindo por outro gráfico desejado.

```
>> subplot(2,1,1),ezplot('sin(x)')
>> subplot(2,1,2),ezplot('exp(x)')
```

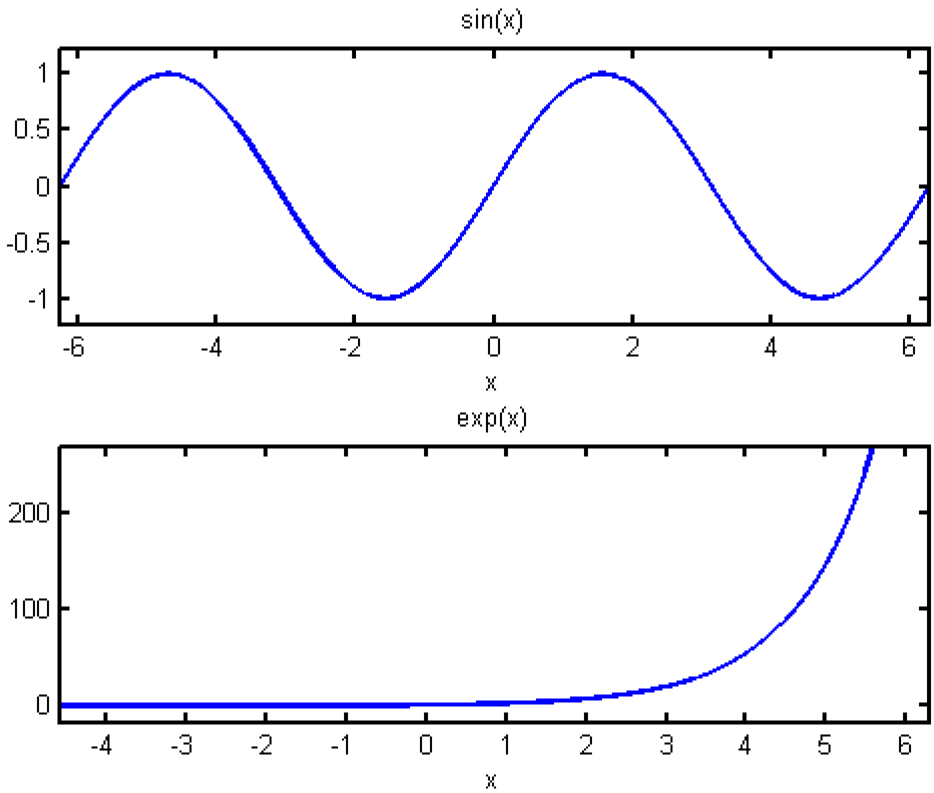


Figura 30 - Janela de gráfico dividida através da função subplot.

Exercício 8 - Plote as funções a seguir com os respectivos comandos e de acordo com cada item: *5sin (x) – plot*; *4sin(x) – stem*.

- **a.** Todas as funções no mesmo gráfico;
- **b.** Cada função em uma janela diferente;
- **c.** Todas as funções na mesma janela, mas em gráficos diferentes.

Exemplo 3 - Criação de arquivo em formato *AVI*. Observe as Fig. 31 e Fig. 33.

```
aviobj=avifile('Filme Seno.avi','fps',50);
hold on;
grid on;
x = -4*pi:0.1:4*pi;
for k = 1:1:size(x,2)-1
    xx = [x(k) x(k+1)];
    yy = [sin(x(k)) sin(x(k+1))];
    h = plot(xx,yy);
    set(h,'EraseMode','xor');
    axis ([-10 10 -1.5 1.5]);
    frame = getframe(gca);
    aviobj = addframe(aviobj,frame);
end
aviobj=close(aviobj);
```

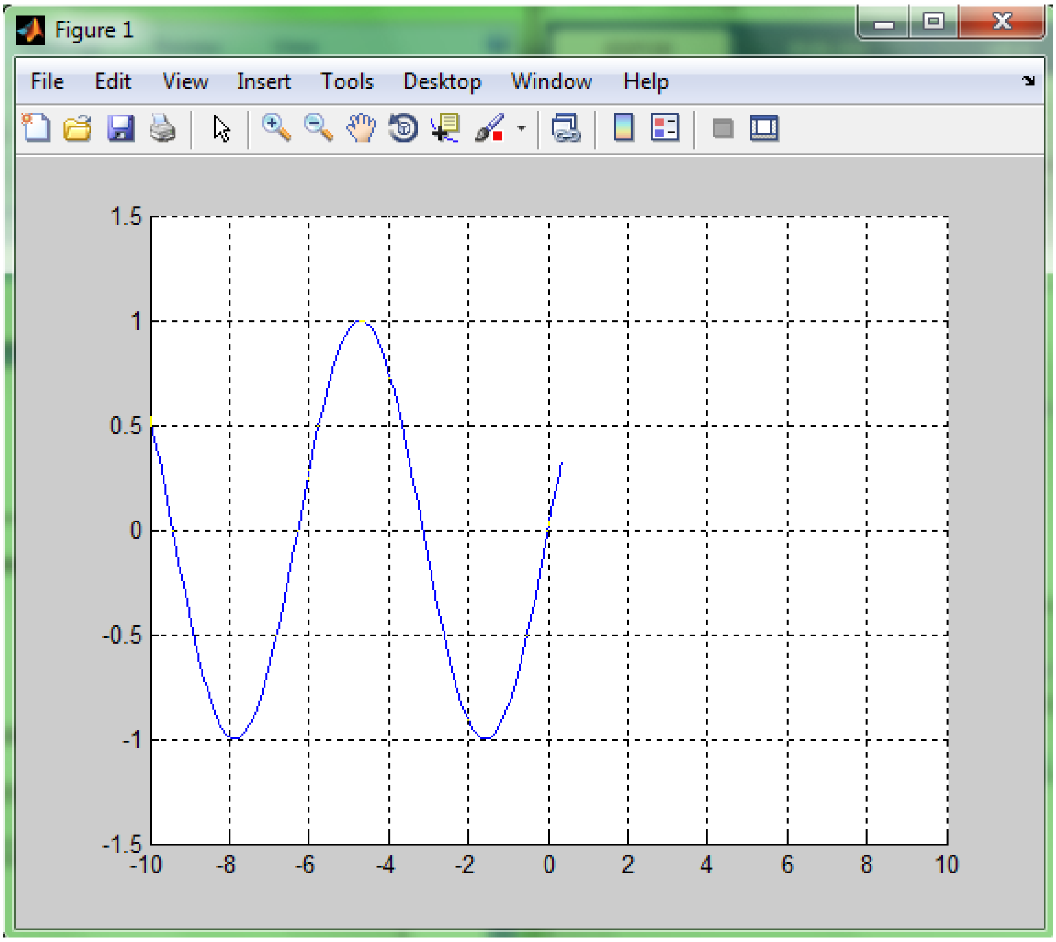


Figura 31 - Janela de criação de arquivo em formato AVI do gráfico *sin(x)*.

```
Aviobj = avifile('Complexo.avi','fps',50);
hold off;
grid on;
t = 0:0.01:4*pi;
x = cos(t);
y = sin(t);
for k = 1:1:length(t)
    c = x(k)+i*y(k);
    h = compass(c);
    set(h,'EraseMode','xor');
    frame = getframe(gca);
    aviobj = addframe(aviobj,frame);
end
aviobj = close(aviobj);
```

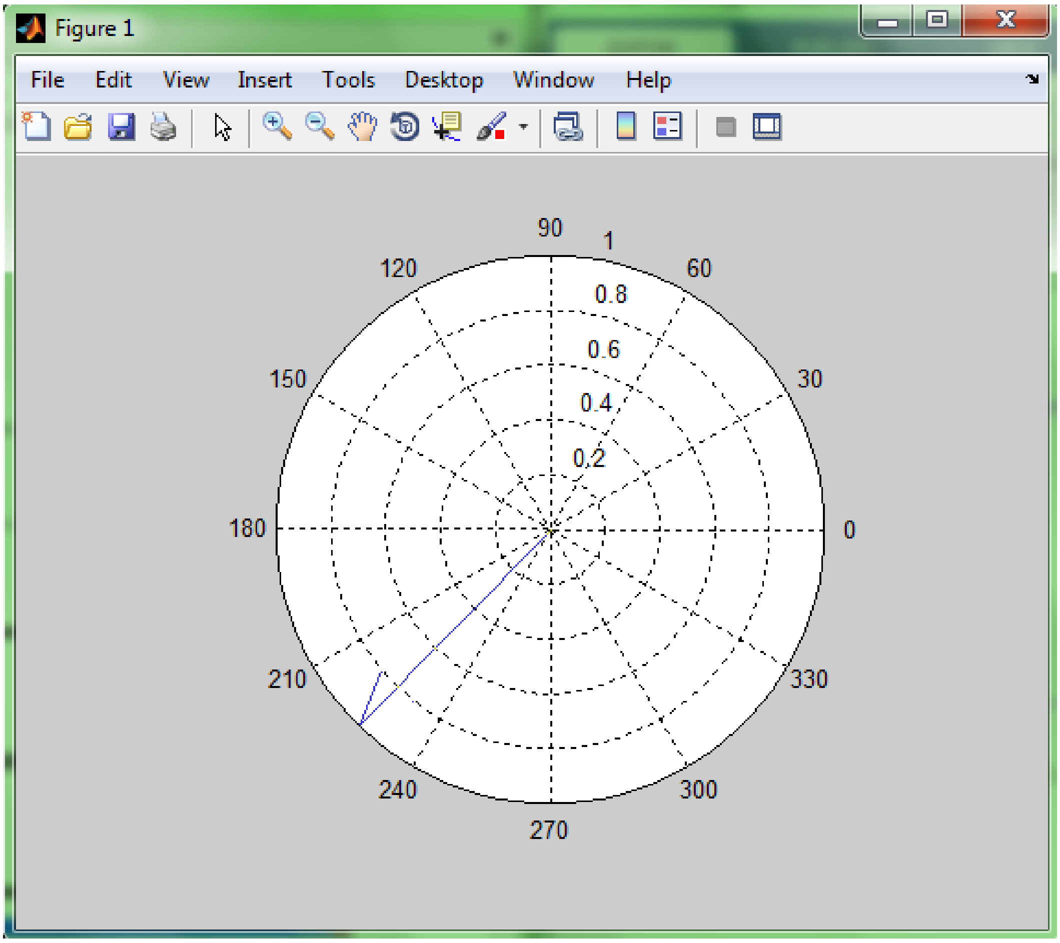


Figura 32 - Janela de criação de arquivo em formato AVI de gráfico polar.

9. MATEMÁTICA SIMBÓLICA

Há, em algumas situações, a necessidade de se trabalhar com variáveis simbolicamente, pois possibilita uma visão mais geral sobre o resultado de um problema. Neste contexto, uma função importante é a *syms*, que declara as variáveis como simbólica. Uma outra função é a *sym*, que transforma uma expressão para a forma literal. Mais detalhes dessas funções são dadas a seguir:

- ***syms***

Definição: Determina que os argumentos acompanhados terão caráter simbólico.

Sintaxe:

syms arg1 arg2 ...

- ***sym***

Definição: Define variáveis, expressões e objetos como simbólicos.

Sintaxe:

S = *sym(A)*

x = *sym('x')*

```
>> rho = sym('(1 + sqrt(5))/2')
%rho =
%5^(1/2)/2 + 1/2
>> syms x y
>> f = x^2*y + 5*x*sqrt(y)
%f =
%x^2*y + 5*x*y^(1/2)
```

Em alguns casos, quando se desejar determinar quais as variáveis simbólicas numa expressão, usa-se a função *findsym*, que retorna os parâmetros que são simbólicos. Uma outra função é a *subs*, que substitui a variável declarada inicialmente simbólica por uma outra ou mesmo por um número. Suas definições estão listadas abaixo:

- ***findsym***

Definição: Determina as variáveis simbólicas em uma expressão.

Sintaxe:

findsym(S)

findsym(S,n)

- ***subs***

Definição: Substituição simbólica em expressão simbólica ou em matriz.

Sintaxe:

R = *subs(S)*

R = *subs(S, new)*

R = *subs(S,old,new)*

R = *subs(S,{old1,old2},{new1,new2})*

Exercício 9 - Dado o procedimento abaixo:

```
y = 3;
w = 30;
syms a b n t x z
f = x^n+y;
g = sin(a*t + b)-cosd(w);
```

Determine os parâmetros que são simbólicos em f e em g. Substitua, em f, x por 2 e y por 3, e em g, a por 4, t por 7 e b por 1.

Exercício 10 - Implemente o método das bisseções sucessivas para a resolução de equações não lineares, Fig. 33.

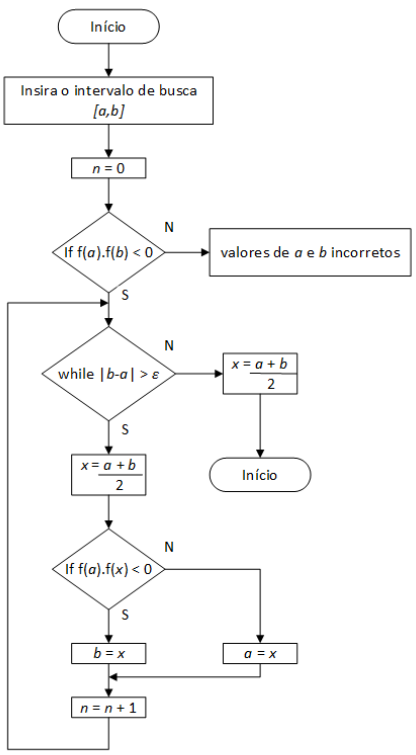


Figura 33 - Fluxograma do Método das Bisseções Sucessivas

10. OPERAÇÕES MATEMÁTICAS BÁSICAS

10.1. Expressões Numéricas

Uma curiosidade é que o MATLAB dispõe de um conjunto de funções que contribuem para a fatoração, expansão, simplificações e entre outros. A Tabela 6 resume bem cada função.

Tabela 6 - Funções de fatração, expansão, simplificação entre outros

Função	Definição
collect	Reescreve a expressão como um polinômio
expand	Expande a expressão em produtos e somas
horner	Determina o fator em comum da expressão
factor	Fatora o polinômio, se os coeficientes são racionais
simplify	Simplifica as expressões, de forma mais geral.
compose	Calcula a composição das funções
finverse	Encontra a inversa funcional da função.

O uso dessas funções é bastante semelhante, por exemplo, dada a expressão:

$$x (x (x - 6) + 11) - 6$$

Para agrupá-la de tal modo que possa ter uma organização em relação ao grau do polinômio, faz:

```
>> syms x
>> collect(x*(x*(x-6)+11)-6)
%ans =
%x^3 - 6*x^2 + 11*x - 6
```

Exercício 11 - Verifique a relação trigonométrica fundamental utilizando a função *simplify*, logo após, determine a forma expandida de *tan(x + y)*.

Exercício 12 - Dado o polinômio $f(x) = 2x^2 + 3x - 5$ e $g(x) = x^2 - x + 7$.

Determine os seguintes polinômios:

a. f(g(x))

b. g(f(x))

10.2. Polinômios

Agora trataremos com os polinômios. Para definir um polinômio no MATLAB, basta entrar com uma matriz linha, nos quais os elementos dela representam os coeficientes do maior para o menor grau.

Por exemplo, o polinômio 5x3 - 9x2 + $\frac{8}{5}$x + $\frac{4}{7}$ é representado como p=[5 -9 8/5 4/7]. É bom lembrar que o polinômio pode ter elementos irracionais como, por exemplo, $\sqrt{2}$ ou π.

As principais funções destinadas para os polinômios são descritas a seguir.

- ***poly***

Definição: Determina os coeficientes do polinômio a partir de suas raízes. Caso a entrada seja uma matriz, este calcula o polinômio característico da matriz.

Sintaxe:

p = *poly(A)*

p = *poly(r)*

```
>> y = [-2 -1];     %Declara um vetor linha [-2 -1]
>> z = poly(y)      %z é o polinômio (x+2)(x+1)=x²+3x+2 que tem como
                    %raízes -2 e -1
%z =
%    1     3     2
>> A = [1 5 3; 0 -2 9; 2 11 -1];      %Declara a matriz
>> poly(A)     %calcula o seu polinômio característico
%ans =
%    1.0000    2.0000 -106.0000   -5.0000
```

- ***roots***

Definição: Retorna um vetor coluna com a(s) raíz(es) do polinômio fornecido.

Sintaxe:

r = *roots(c)*

```
>> c = [1 3 2];    %declara um vetor correspondente ao polinômio
                   %x²+3x+2
>> x = roots(c)    %Calcula as raízes desse polinômio, que são e -1
                   %Observe a oposição entre as funções roots e poly
%x =
%   -2
%   -1
```

- ***polyval***

Definição: Determina o valor do polinômio para uma determinada entrada. Se a entrada for uma matriz, a função retorna o valor do polinômio para cada elemento.

Sintaxe:

y = *polyval(p,X)* → y receberá os valores do polinômio desenvolvido para cada elemento da matriz X.

```
>> polinomio = [1 3 2];    % polinômio=x4+5x3-2x2+8x+3.2
>> a = [1 -1; 3 2.83];
>> valores=polyval(polinomio,a)
%valores =
%  15.2000 -10.8000
% 225.2000 187.2906
%valores(1,1) = a(1,1)4+5a(1,1)3-2 a(1,1)2+8 a(1,1)+3.2
```

- ***polyfit***

Definição: Determina o polinômio interpolador com os pontos dados por x e y com o grau n. Os coeficientes são retornados numa matriz linha.

Sintaxe:

p = *polyfit(x,y,n)*

```
>> x=[0:0.1:2.5];
>> y = sqrt(x);
>> polinomio_interpolador = polyfit(x,y,3)
>> pontos_interpoladores = polyval(polinomio_interpolador,x)
>> plot(x,y,'color','r')
>> hold on
>> plot(x,pontos_interpoladores)
```

Exercício 13 - São dados os pontos (1;-1), (2;-7), (5;-8) e (8;10).

Exercício 14 - Determine o polinômio que interpola estes pontos;

Exercício 15 - Calcule as suas raízes e o esboce em um gráfico;

Exercício 16 - Destaque o ponto no qual se tem o valor do polinômio para x = 3.

10.3. Solucionando Equações ou Sistemas

Quando você tiver um emaranhado de equações, resultando em um sistema, o MATLAB poderá ser uma ótima solução. Ao utilizar a função *solve*, você será capaz de economizar tempo e evitar resolver um tedioso sistema braçalmente. A declaração desta função segue abaixo:

- **solve**

Definição: Determina o valor do polinômio para uma determinada entrada. Quando a solução é armazenada em uma variável, o retorno é dado em uma estrutura de dados.

Sintaxe:

solve(eq) → Resolve a equação eq=0

solve(eq,var) → Determina as soluções de eq=0, em função da variável *var*.

solve(eq1,eq2,...,eqn) → Resolve um sistema de equações definidas.

g = *solve(eq1,eq2,...,eqn,var1,var2,...,varn)* → Calcula as soluções de um sistema de soluções em função das variáveis pré-definidas.

Partindo para um âmbito mais complexo, quando se trata de equações diferenciais, a função destinada para este caso é a *dsolve*, definida abaixo:

- ***dsolve***

Definição: Soluciona simbolicamente uma equação ou sistema de equações diferenciais ordinárias.

Sintaxe:

r = *dsolve('eq1,eq2,...', 'cond1,cond2,...','v')*

r = *dsolve('eq1','eq2',...,'cond1','cond2',...,'v')*

dsolve('eq1,eq2,...','cond1,cond2,...', 'v')

Exemplo 4 - Determine a solução de $\frac{dx}{dt}$ = - ax .

```
>> dsolve('Dx = -a*x')
```

Exercício 17 - Eu tinha o triplo da idade que tu tinhas, quando eu tinha a idade que tu tens. Quando tu tiveres a minha idade, a diferença de nossas idades será de duas décadas. Determine nossas idades utilizando a função *solve*.

Exercício 18 - Sabe que a aceleração de um carro em uma estrada é a = -4 x, em que x representa a posição no instante t. Determine a posição no instante π, sabendo que este carro parte, no instante 0, do ponto 1, sendo que o motorista parou instantaneamente enquanto estava em $\frac{\pi}{2}$. Considere todas as unidades no S.I.

11. CÁLCULO DIFERENCIAL

O MATLAB disponibiliza funções que facilitam a operação de certos cálculos que são difíceis para o usuário. Por exemplo, a função *diff()*, *int()* e *limit()* são algumas delas, nas quais diferenciam, integram ou calculam o limite de uma função de acordo com os parâmetros oferecidos, respectivamente.

Vejamos essas e outras funções a seguir:

11.1. Limites

- ***limit***

Definição: Determina o limite de uma expressão simbólica

Sintaxe:

limit(F,x,a) → calcula o limite de uma expressão simbólica F com x tendendo a a;

limit(F,a) → determina o limite de F com uma variável simbólica tendendo a a;

limit(F) → determina o limite com a = 0 como default;

limit(F,x,a,'right') → calcula o limite com x tendendo a a pela direita;

limit(F,x,a,'left') → calcula o limite com x tendendo a a pela esquerda;

Exemplo 5 - Faça o seguinte limite pelo MATLAB: $\lim_{x\to 1^+} \frac{|x^2|-1}{x^2-1}$

```
>> limit('(abs(x^2)-1)/(x^2-1)',x,1,'right')
%ans =
%    1
```

11.2. Diferenciação

- ***diff***

Definição: Calcula a diferencial de uma função/matriz dada.

Sintaxe:

diff(S) → diferencia a expressão simbólica S em função de uma variável simbólica;

diff(S,'v') → diferencia S em torno de uma variável simbólica v;

diff(S,n) ⟶ diferencia, para um n inteiro positivo, S por n vezes;

diff(S,'v',n) ⟶ diferencia em torno de uma variável v, S por n vezes.

Exemplo 6 - Para determinar a derivada de 1ª ordem de f(x) = $\sqrt{\ln(x) + e^x}$, faz-se:

```
>> syms x;
>> f = sqrt(log(x)+exp(x));
>> diff(f)
%ans =
%(exp(x) + 1/x)/(2*(exp(x) + log(x))^(1/2))
>> pretty(ans)
```

11.3. Integração

- ***int***

Definição: Calcula integral de uma função simbólica dada.

Sintaxe:

int(S) ⟶ integração indefinida a função S em respeito a uma variável simbólica já definida;

int(S,a,b) ⟶ integra de forma definida a função S de a a b;

int(S,v,a,b) ⟶ integra de a a b em função de uma variável v;

Exemplo 7 - Dado a função f (x) = $\sqrt{x^2 + 5}$, calcule a integral:

- **Indefinida**

```
>> syms x
>> y=sqrt(x^2+5);
>> f=int(y,x)
%f =
%(5*asinh((5^(1/2)*x)/5))/2 + (x*(x^2 + 5)^(1/2))/2
```

- **Definida de 2 a 5**

```
>> g = int(y,x,2,5)
%g =
%(5*log(5^(1/2) + 6^(1/2)))/2 - (5*log(5))/4 + (5*30^(1/2))/2 - 3
```

Caso a visualização de f não seja satisfatória, usa-se a função *pretty*, que transforma a saída de acordo com a representação matemática, conforme ilustra abaixo:

```
>> pretty(g)
```

Como a integral é calculada de forma simbólica, não é calculado o valor numérico da função g. Entretanto, o MATLAB não deixa a desejar neste ponto, como ao utilizar a função *eval*. Por exemplo, fazendo a instrução abaixo, você obtém:

```
>> eval(g)
```

11.4. Integrais definidas pela Regra Trapezoidal

É um método utilizado quando a área sob uma curva é representada por trapézios entre um intervalo [a,b] pré-definido, sendo o número de divisões n. Em representação matemática, tem-se:

$$\int_a^b f(x)dx = \frac{b-a}{2n}[f(x_0) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(x_n)]$$

em que x_i representa o ponto no final de cada trapézio, sendo $x_0 = a$ e $x_n = b$. No MATLAB a função que possibilita integração a partir deste método é o *trapz*, definida abaixo:

- ***trapz***

Definição: Determina a integração de uma função a partir da Regra do Trapézio.

Sintaxe:

Z = *trapz(Y)*

Z = *trapz(X,Y)*

11.5. Integrais definidas pela Regra de Simpson

O método de Simpson é baseado, dado três pontos sobre a curva da função, na aproximação desses pontos em uma parábola. Então, tomados n subintervalos, onde n é par, e cuja extremidade da curva é delimitada por f (a) e por f (b) , logo, a integral de uma função é denotada por:

$$\int_a^b f(x)dx = \frac{b-a}{3n}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]$$

A maioria das calculadoras programadas utiliza esta regra, que é mais utilizada em termos computacionais. No MATLAB, a função encarregada para esse fim é a *quad*, mostrada abaixo:

- ***quad***

Definição: Determina a integração de uma função a partir da Regra de Simpson.

Sintaxe:

q = *quad(fun,a,b)*

q = *quad(fun,a,b,tool)* → *tool* corresponde o erro que a integral retornará, sendo o *default* de 10^{-3}.

É bom destacar que *fun* deve ser uma função do tipo arquivo.m. Por exemplo, para calcular a integral de y = e-x no intervalo de 0 a 3, faz-se o seguinte:

Primeiro, se cria o arquivo .m correspondente à função que deseja integrar, ou seja:

```
function y=funcao(x)
y=exp(-x^2);
```

Em seguida, basta utilizar a função *quad*, conforme modelo abaixo:

```
>> quad('funcao',0,3)
%ans =
%     0.8862
```

11.6. Integração Dupla

O MATLAB possui a função *dblquad* que determina a integral dupla de uma função, conforme definição abaixo:

- ***dblquad***

Definição: Determina a integração dupla de uma função.

Sintaxe:

q = *dblquad(fun,xmin,xmax,ymin,ymax)*

q = *dblquad(fun,xmin,xmax,ymin,ymax,tol)* → *tol* é a precisão que deseja calcular, sendo o *default* 10^{-6}.

11.7. Integração Tripla

Também é possível calcular a integral tripla de uma função no MATLAB, utilizando neste caso a função *triplequad*, cuja definição segue abaixo:

- ***triplequad***

Definição: Determina a integração tripla de uma função.

Sintaxe:

triplequad(fun,xmin,xmax,ymin,ymax,zmin,zmax)

triplequad(fun,xmin,xmax,ymin,ymax,zmin,zmax,tol) → *tol* é a precisão que deseja calcular, sendo o *default* 10^{-6} .

11.8. Outras funções

- ***gradient***

Definição: Determina o gradiente numericamente.

Sintaxe:

FX = *gradient(F)* → retorna o coeficiente do gradiente de F em relação a ∂x(*default*).

[FX,FY,FZ,...] = *gradient(F)* → retorna a matriz com os valores de ∂x, ∂y, ∂z..., respectivamente.

- ***divergence***

Definição: Determina o divergente de um campo vetorial.

Sintaxe:

div = *divergence(X,Y,Z,U,V,W)* → determina o divergente do campo vetorial 3D U, V e W. X, Y e Z definem os limites de U, V e W, respectivamente.

div = *divergence(X,Y,U,V)* → calcula agora para 2D.

div = *divergence(U,V,W)* → usa como *default* para X, Y e Z o valor de *meshgrid*(1:n,1:m,1:p).

Exemplo 8 - Um exemplo clássico para o uso de gradiente seria na determinação do Campo Elétrico devido ao efeito de uma carga pontual de 18 ηC. O código segue abaixo.

```
>> [x,y,z] = meshgrid(-1:0.3:1);
>> V = (18e-12)./(4.*pi.*8.85e-12.*sqrt(x.^2+y.^2+z.^2));
>> [px,py,pz] = gradient(V,0.3,0.3,0.3);
>> Ex = (-1).*px;
>> Ey = (-1).*py;
>> Ez = (-1).*pz;
>> quiver3(x,y,z,Ex,Ey,Ez)
>> axis([-1 1 -1 1 -1 1])
```

Veja que a função *quiver* plota o que representaria o campo elétricodevido à carga, Fig. 34.

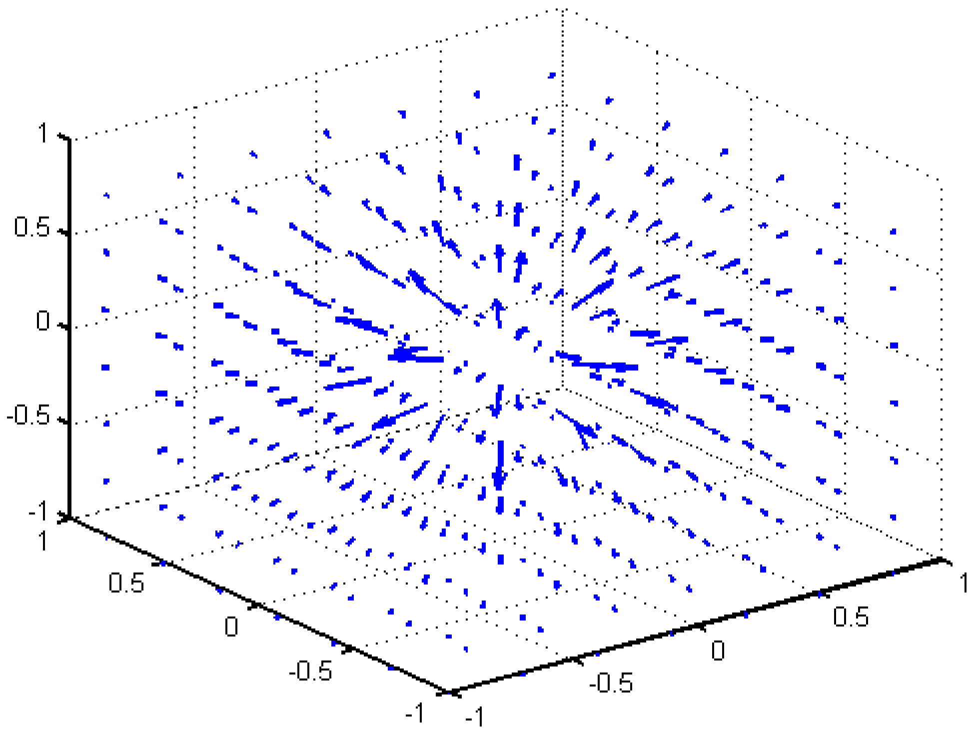


Figura 34 - Campo elétrico pela função *quiver*

12. SÉRIES NUMÉRICAS

12.1. Somatório

Uma função muito utilizada em Séries Numéricas é a *symsum*, que encontra o somatório simbólico de uma expressão. A sua descrição segue abaixo:

- ***symsum***

Definição: Determina o somatório simbólico de uma expressão.

Sintaxe:

r = *symsum(s)* → encontra o somatório da função s em função de uma variável simbólica pré-definida.

r = *symsum(s,v)* → fornece o somatório em função da variável v.

r = *symsum(s,a,b)* → determina o somatório de s variando a incógnita de a até b.

r = *symsum(s,v,a,b)* → determina o somatório de s variando a incógnita v de a até b.

Exercício 19 - Determine os seguintes somatórios:

a. $\sum_{0}^{x-1} x^2$

b. $\sum_{1}^{\infty} \frac{1}{(2n-1)^2}$

12.2. Série de Taylor

A Série de Taylor é definida como sendo:

$$P(x) = \sum_{n=0}^{\infty} (x - x_0)^n . \frac{f^{(n)}(x_0)}{n!}$$

Considerando P(x) como sendo o polinômio de *Taylor*, de ordem n, em torno do ponto x_0, então P(x) é o único polinômio de grau no máximo n que aproxima localmente f em volta de x0 de modo que o erro E(x) tenda a zero mais rapidamente que $(x\text{-}x0)^n$, quando x → x_0.

O MATLAB dispõe da função *taylor* , conforme pode ser visto abaixo:

- ***taylor***

Definição: Expande em série de *Taylor*.

Sintaxe:

taylor(f) → faz a aproximação pelo polinômio de *Taylor* até a quinta ordem para a função simbólica f.

taylor(f,n,v) → retorna o polinômio de *Taylor* para a função simbólica f até o grau n-1 para a variável especificada por v.

taylor(f,n,v,a) → retorna a aproximação de *Taylor* de f em torno do ponto a, que pode ser simbólica ou um valor numérico.

Por exemplo, calculando o polinômio de *Taylor* para a função $f(x) = \frac{1}{5+4.\cos(x)}$, tem-se

```
>> syms x
>> f = 1/(5+4*cos(x))
%f =
%1/(5+4*cos(x))
>> T = taylor(f)
%T =
%1/9+2/81*x^2+5/1458*x^4+49/131220*x^6
>> pretty(T)
```

Exercício 20 - Determine, pelo polinômio de *Taylor* de ordem 2, o valor aproximado de $\sqrt[3]{8,2}$

13. ANÁLISE DE SINAIS

13.1. Transformação de Variável Independente

x(t) → x(αt + β)

|α| < 1 → Expansão

|α| > 1 → Compressão

α < 0 → Reflexão

β ≠ 0 → Deslocamento

- **Deslocamento no tempo:**

```
>> clear, clc, clf
>> x = -2:6;
>> y = 2*x;
>> n0 = input('Digite o valor de deslocamento, n0= ');
>> subplot(2,1,1), stem(x,y), grid on, xlim([-20 20])
>> hold on
>> xnovo = x+n0;
>> subplot(2,1,2), stem(xnovo,y), grid on, xlim([-20 20])
>> hold off
```

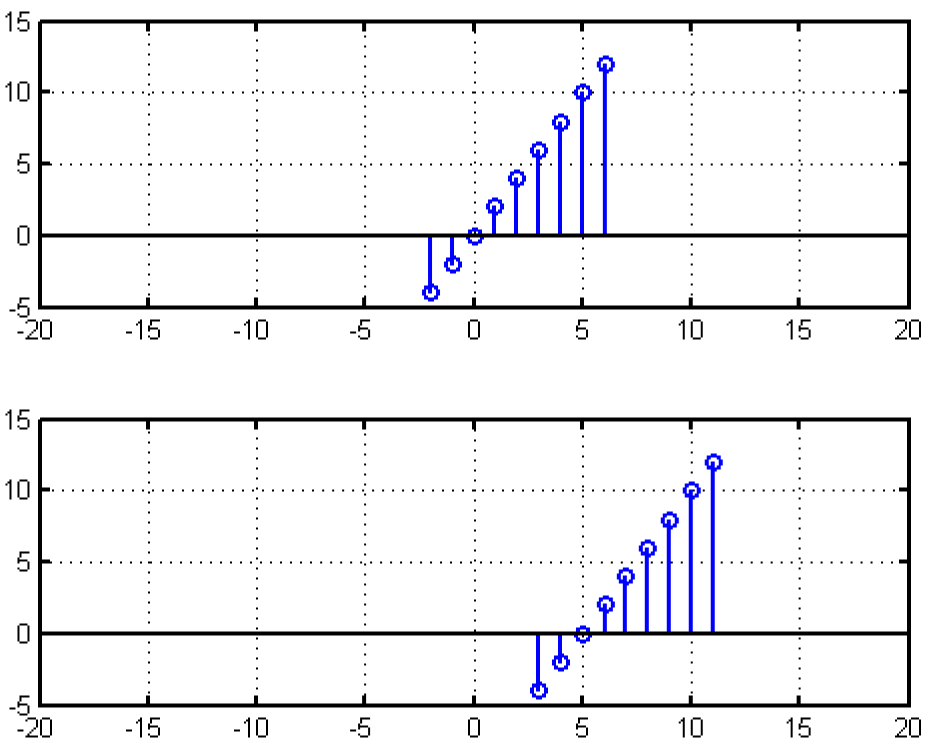


Figura 35 - Propriedade de deslocamento no tempo.

- **Reflexão:**

```
>> clear, clc, clf
>> x = -2:8;
>> y = x;
>> subplot(2,1,1), stem(x,y), grid on, xlim([-20 20])
>> hold on
>> xl = -x;
>> subplot(2,1,2), stem(xl,y), grid on, xlim([-20 20])
>> hold off
```

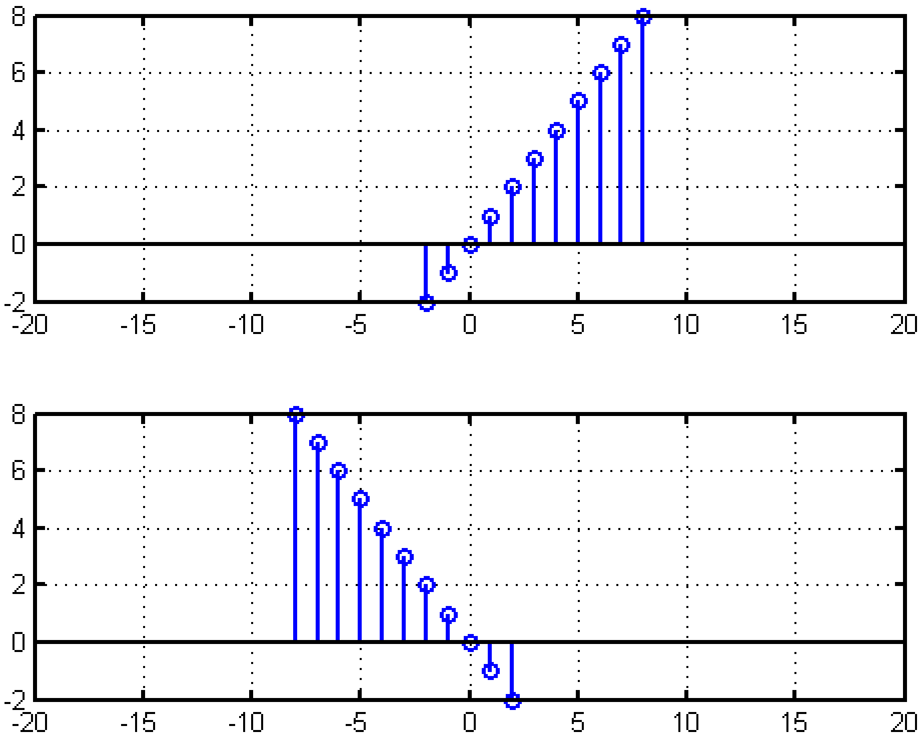


Figura 36 - Propriedade de reflexão.

- **Escalonamento:**

```
>> clear, clc, clf
>> x = -2:6;
>> y = 2*x;
>> a = input(' Digite o valor de escalonamento, a= ')
>> subplot(2,1,1), stem(x,y), grid on, xlim([-20 20])
>> hold on
>> xl = x/a;
>> subplot(2,1,2), stem(xl,y), grid on, xlim([-20 20])
>> hold off
```

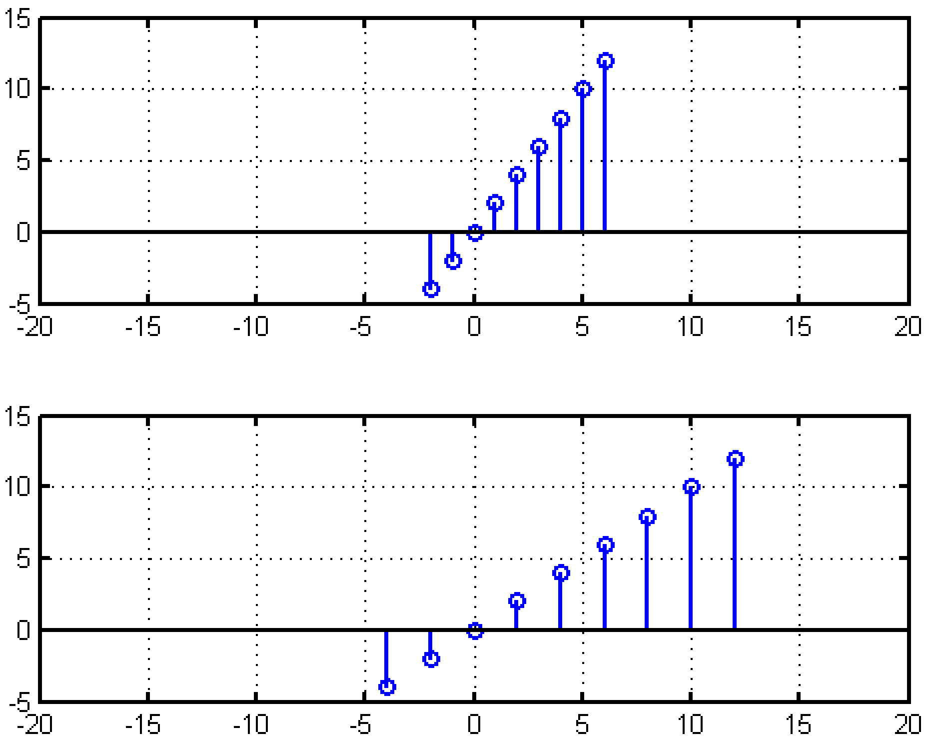


Figura 37 - Propriedade de escalonamento.

13.2. Funções Pré-definidas

- **Impulso**

δ [n] = 0, n $\neq$ 0

1, n $\neq$ 0

δ (t) = 0, t $\neq$ 0

1, t $\neq$ 0

```
function [u] = impulso(n,N)
for k = 1:length(n)
    if n(k)~=N
        u(k) = 0;
    else
        u(k) = 1;
    end
end
```

Command Window:

```
>> x = -2:7;
>> y = impulso(x,3);
>> stem(x,y)
```

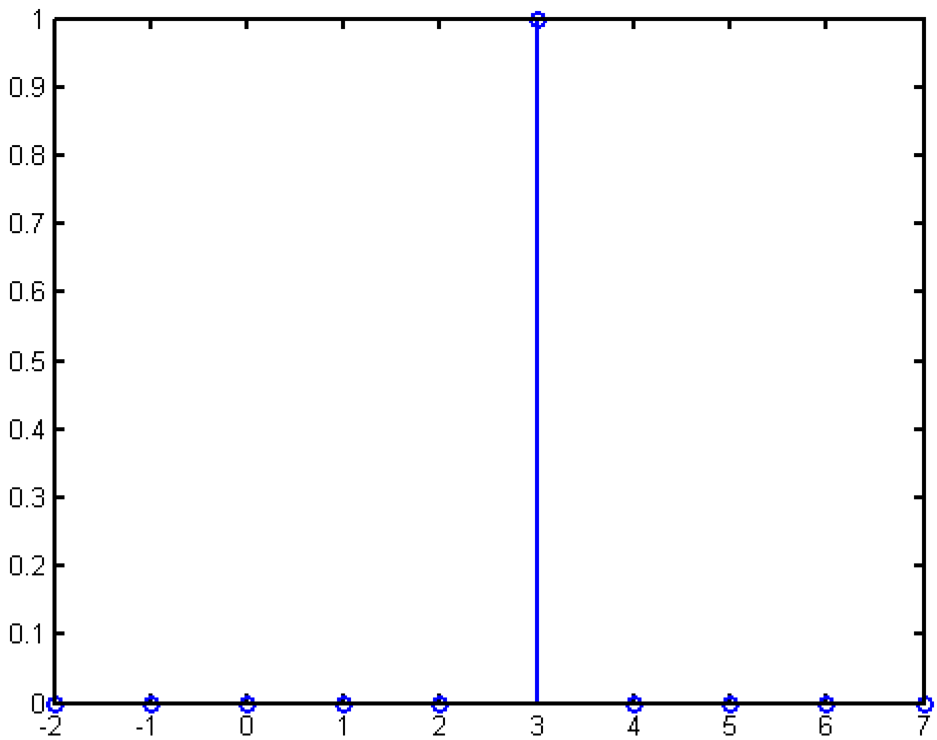


Figura 38 - Função impulso.

- **Degrau**

u [n] = 0, n < 0

1, n ≥ 0

u (t) = 0, t < 0

1, t ≥ 0

```
function [u] = degrau(n,N)
for k = 1:length(n)
    if n(k)<N
        u(k) = 0;
    else
        u(k) = 1;
    end
end
```

Command Window:

```
>> n = -2:7;
>> y = degrau(n,3);
>> stem(n,y)
```

Exercício 21 - Determine:

a. y[n] = u[n+10] - 2u[n+5] + u[n-6]

```
>> n = -20:20;
>> y = degrau(n,-10)-2*degrau(n,-5)+degrau(n,6);
>> stem(n,y)
```

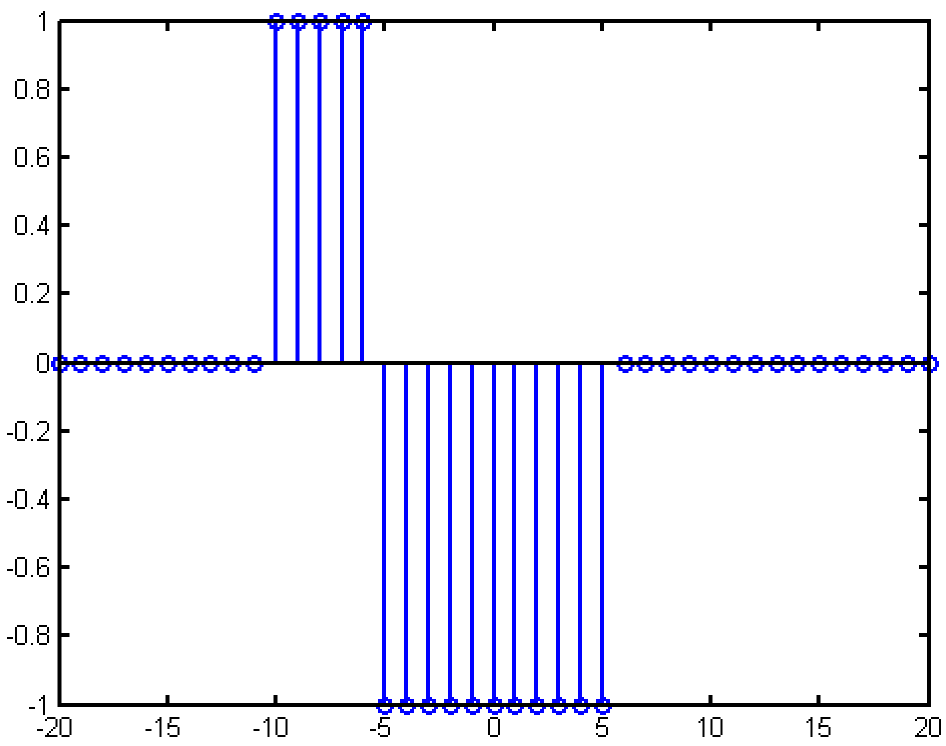


Figura 39 - Gráfico do item *a*.

b. y[n] = $\left(\frac{1}{2}\right)^n$ - u[n-3]

```
>> n = -20:20;
>> y = ((1/2).^(n)).*degrau(n,3);
>> stem(n,y)
```

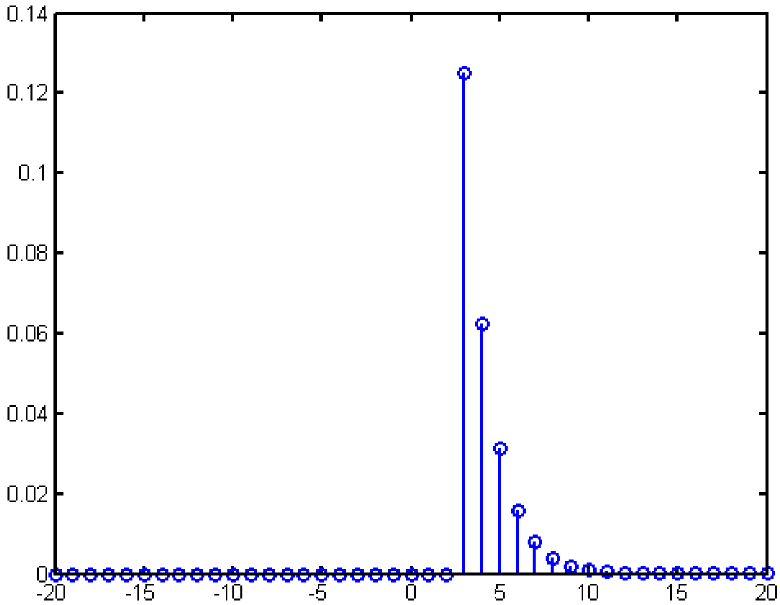


Figura 40 - Gráfico do item *b*.

c. y[n] = cos(0,5πn) - u[n-4]

```
>> n = -20:20;
>> y = cos(0.5*pi*n).*degrau(n,4);
>> stem(n,y)
```

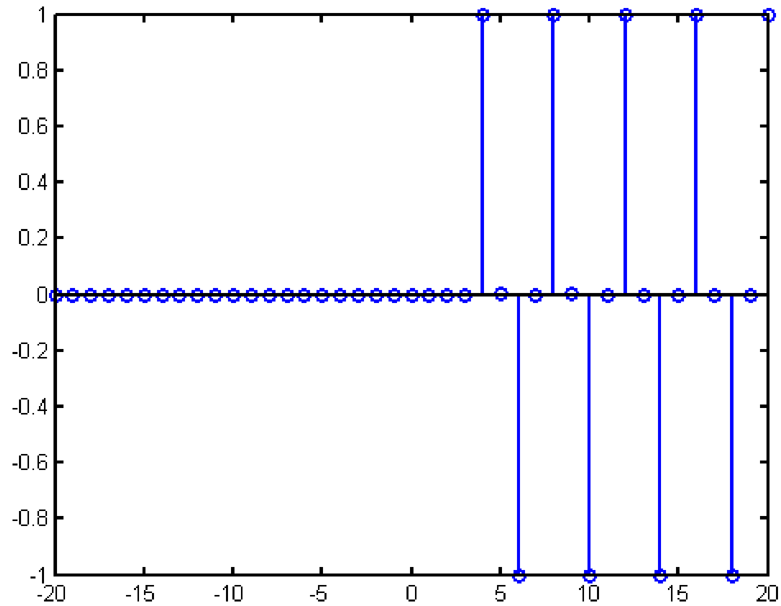


Figura 41 - Gráfico do item *c*.

d. y[n] = $e^{2sen(n)}$

```
>> n = -20:20;
>> y = exp(2*(sin(n)));
>> stem(n,y)
```

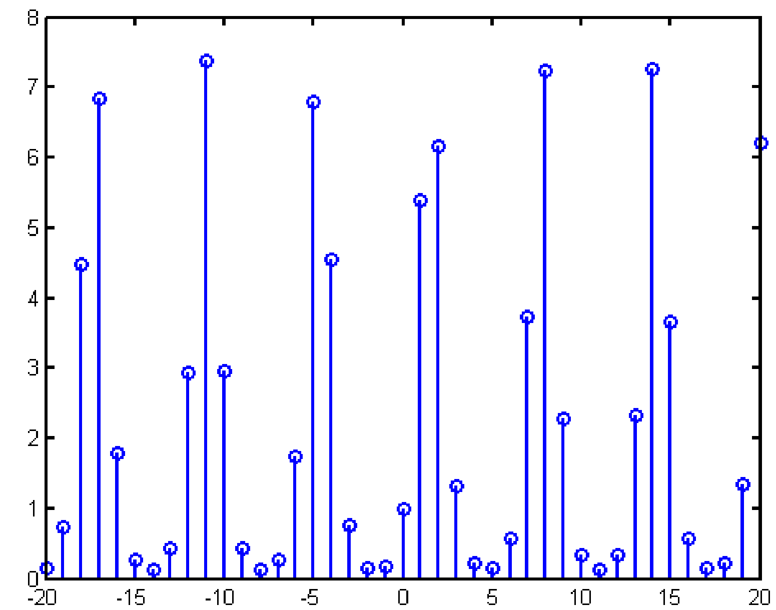


Figura 42 - Gráfico do item *d*.

e. y[n] = $sen\left(\frac{n}{2}\right)$

```
>> x = -20:1:20;
>> y = sinc(x/2);
>> stem(x,y)
```

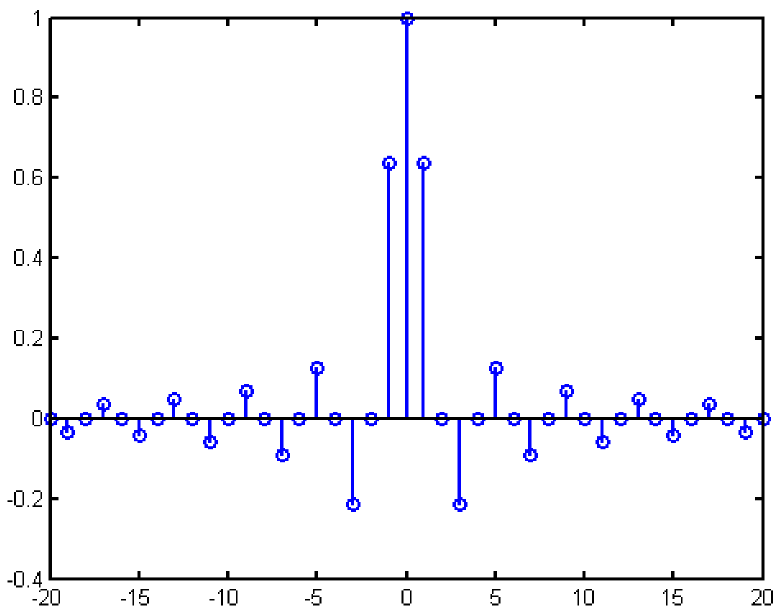


Figura 43 - Gráfico do item *e*.

É bom salientar que nos exemplos anteriores foram dados exemplos de programas no qual se obtém as funções impulso e degrau. Entretanto, o MATLAB também possui funções que possibilitam isso de forma mais rápida, que são as funções *dirac* e a *heaviside*, conforme veremos a seguir:

- ***dirac***

Definição: Obtém a função delta de *dirac*, ou seja, a função impulso no intervalo determinado por x.

Sintaxe:

dirac(x)

```
>> x = -10:10;
>> y = dirac(x-5); %Impulso no instante 5
>> stem(x,y)
```

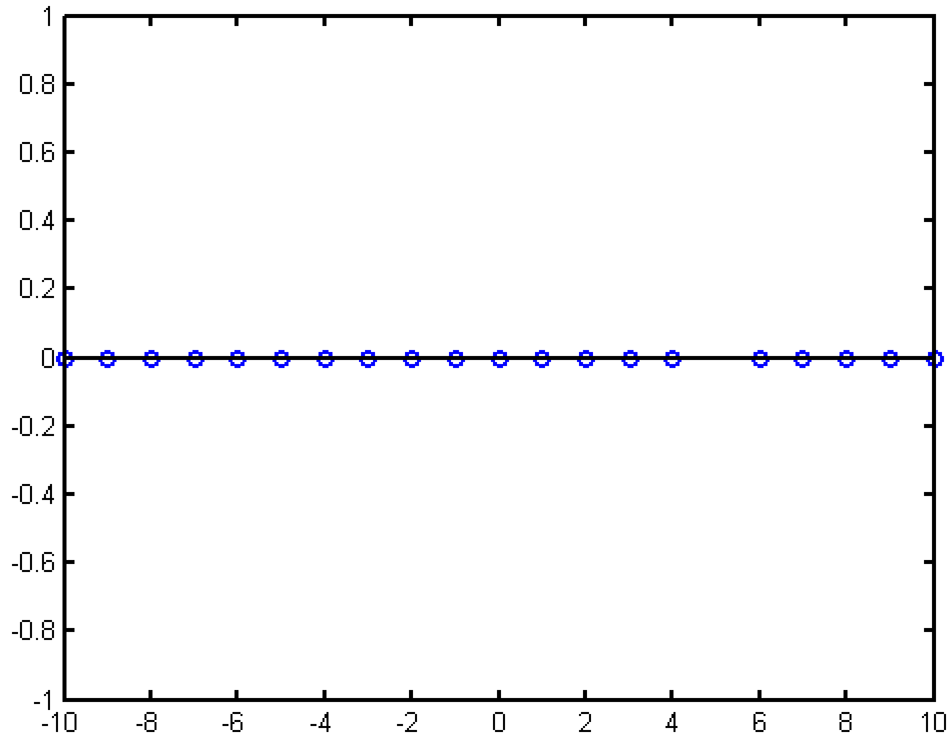


Figura 44 - Impulso com a função *dirac*.

- ***heaviside***

Definição: Determina a função degrau no intervalo determinado por x.

Sintaxe:

heaviside(x)

```
>> x = -10:10;
>> y = heaviside(x-5); %Degrau no instante 5
>> stem(x,y)
```

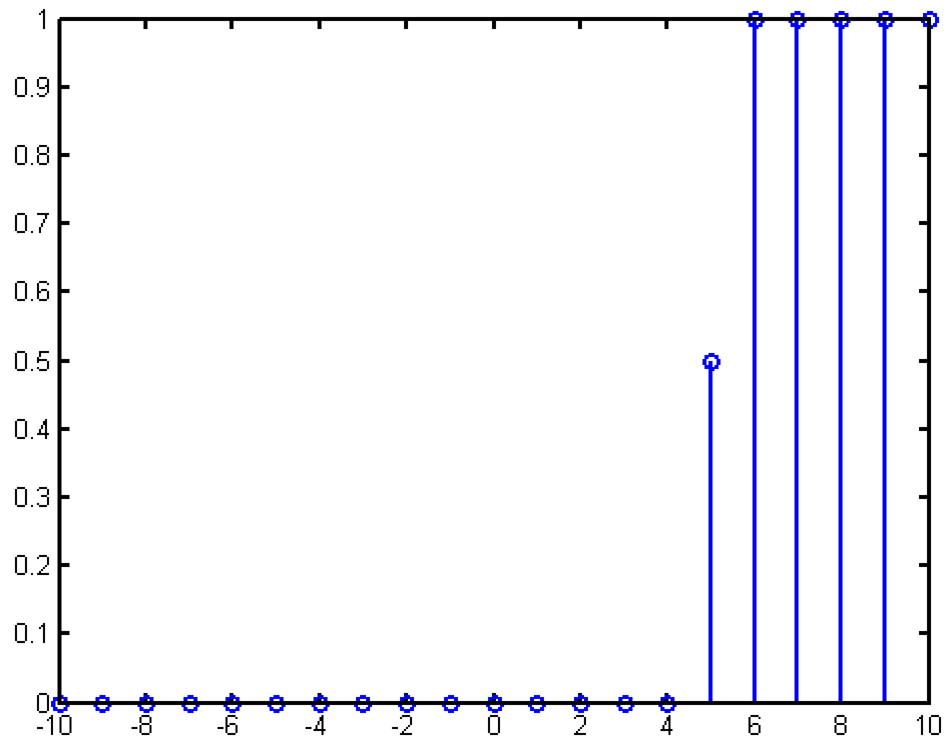


Figura 45 - Degrau com a função *heaviside*.

Exemplo 9 - Sabe-se que a função impulso é a derivada da função degrau. Determina este fato utilizando o MATLAB.

```
>> syms x
>> diff(heaviside(x),x)
%ans =
%dirac(x)
```

Exercício 22 - Verifique a integral de sen(x). δ (x-5)

13.3. Convolução

A convolução é uma ferramenta matemática que expressa a saída de um sistema de tempo, seja este discreto ou contínuo, em função de uma entrada pré-definida e da resposta ao impulso do sistema.

O MATLAB possui uma função chamada de conv que realiza a convolução de sinais de duração finita. Por exemplo, sejam dois vetores x e h representando sinais. O comando y = *conv*(x, h) gera um vetor y que denota a convolução dos sinais x e y.

Veja que o número de elementos em y é dado pela soma do número de elementos em x e y menos um, devido ao processo de convolução. O vetor ny dado pelo espaço de tempo tomado pela convolução é definido pelo intervalo entre a soma dos primeiros elementos de nx e nh e a soma dos últimos elementos de nx e nh, sendo nx o espaço tempo definido para o vetor x e nh o espaço de tempo definido para o vetor h.

ny = [(*min(nx)* + *min(nh)*):(*max(nx)* +*max(nh)*)];

Vejamos a sintaxe de *conv* abaixo:

- ***conv***

Definição: Determina a convolucão de dois sinais ou a multiplicação de dois polinômios.

Sintaxe:

w = *conv*(u,v)

```
>> h = [1 2 1];
>> x = [2 3 -2];
>> y = conv(x,h)
%y =
%    2     7     6    -1    -2
```

Exemplo 10 - Determine os coeficientes do polinômio obtido ao multiplicar os polinômios $3x^2 + 3x$ com $2x + 2$.

```
>> a = [3 3 0];
>> b = [2 2];
>> y = conv(a,b);
%y =
%    6    12     6     0
```

Logo, o polinômio obtido seria $6x^3 + 12x^2 + 6x$.

Exemplo 11 - Determine a resposta de um sistema com a entrada x[n] = u[n - 2] - u[n - 7] , sabendo que a resposta desse sistema ao impulso é h[n] = u[n] - u[n - 10] .

```
>> h = ones(1,10);
>> x = ones(1,5);
>> n = 2:15;
>> y = conv(x,h);
>> stem(n,y);
```

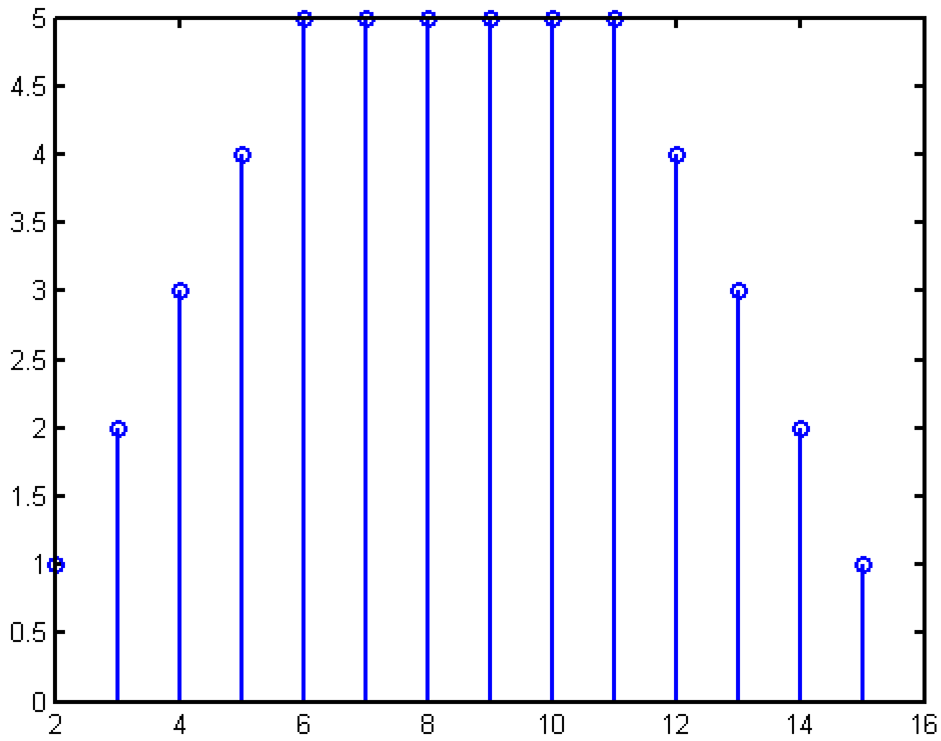


Figura 46 - Gráfico de convolução.

Exercício 23 - Use o MATLAB para determinar a saída do sistema com entrada x[n] = 2{u[n + 2] - u[n - 12]} sabendo que a resposta ao impulso desse sistema é h[n] = $0{,}9^n$ {u[n - 2] - u[n - 13]}.

13.4. Equações de Diferenças

As Equações de Diferenças é uma forma de expressarmos um sistema na forma recursiva que permita que a saída do sistema fosse computada a partir do sinal de entrada e das saídas passadas.

Um comando que é possível realizar uma função similar seria o *filter*, definida a seguir:

- ***filter***

Definição: Expressa a descrição em equação de diferenças de um sistema em uma forma recursiva que permita que a saída do sistema seja computada a partir do sinal de entrada e das saídas passadas.

Sintaxe:

y = *filter(b,a,X)*

y = *filter(b,a,X,zi)*

Veja acima que apareceu o parâmetro zi, que determina a condição inicial de y. Este zi é uma matriz com as condições iniciais, sendo os valores passados de y.

Exemplo 12 - Um exemplo clássico no uso de *filter* é determinar a seqüência de Fibonacci, definida como o número atual ser igual a soma dos dois números anteriores. Em linguagem matemática, tem-se y[n] - y[n - 1] - y[n - 2] = 0 em que y é a saída do sistema.

Veja que ele não depende de uma entrada, mas ao usarmos o filter, é necessário usar a entrada apenas para definir o número de elementos da seqüência no qual se deseja obter, assim como é um parâmetro indispensável para o uso da função *filter*.

Será dado como condição inicial a matriz [1 0], correspondente a entrada não desejada y[-1] e y[-2], indispensável para obter os outros valores.

O código do programa que pode ser implementado no M-file segue abaixo. Neste caso, se deseja adquirir 20 valores.

```
>> a = [1, -1, -1];
>> b = [0];
>> x = ones(1,20);
>> y = filter(b,a,x,[1 0]);
```

Exercício 24 - Determine, utilizando *filter*, a seqüência de Fibonacci.

Quando se trabalha com sistemas de equações de diferenças, no qual precisa determinar a resposta desse sistema ao impulso, o comando *impz* se torna bastante útil. A sua sintaxe segue abaixo:

- ***impz***

Definição: Determina a resposta ao impulso de um sistema de equações de diferenças.

Sintaxe:

[h,t] = *impz(b,a)*

[h,t] = *impz(b,a,n)*

O comando [h,t] = *impz(b,a,n)* avalia n valores da resposta ao impulso de um sistema descrito por uma equação de diferenças. Os coeficientes da equação de diferenças estão contidos nos vetores b e a no que se refere a *filter*. O vetor h contém os valores da resposta ao impulso e t contém os índices de tempo correspondentes.

13.5. FFT (Transformada Rápida de *Fourier*)

A Transformada de *Fourier* leva uma função no domínio do tempo para o domínio da freqüência, no qual podemos analisar as freqüências mais importantes (com maior amplitude) de uma função. A transformada inversa de *Fourier* faz o processo inverso, passa uma função do domínio da frequência para o domínio do tempo.

A Transformada de *Fourier* e sua inversa podem ser calculadas a partir das expressões abaixo, respectivamente:

$$F(\omega) = \int_{-\infty}^{\infty} S(t).e^{-j\omega t}dt$$

$$S(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega).e^{j\omega t}d\omega$$

Onde ω é a freqüência fundamental.

A FFT (Transformada rápida de *Fourier*) é um algoritmo computacional otimizado que calcula a Transformada Discreta de *Fourier* mais rapidamente. A FFT também pode servir de aproximação para a Transformada de Tempo Discreto de *Fourier*, Série de *Fourier* e a própria Transformada de *Fourier*.

Uma propriedade da Transformada de *Fourier* é que a transformada da convolução de duas funções equivale à multiplicação das duas no domínio da freqüência. Portanto para calcular a convolução de uma função levamos os dois sinais para o domínio da freqüência, multiplicamos e voltamos para o domínio do tempo. Veja a expressão abaixo:

y(t) = x(t) * h(t) = IFFT[FFT(x(t)).FFT(h(t))]

Exemplo 13 - Dado o circuito RC da Fig. 47, determine a resposta ao impulso e a corrente no capacitor iC (t) quando a entrada x(t) é igual a e^{-t}.

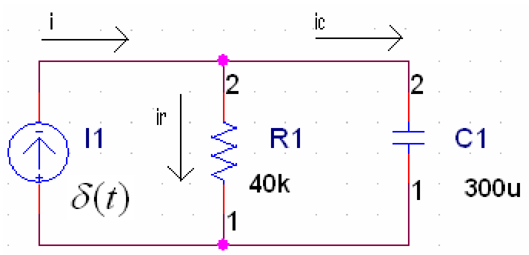


Figura 47 - Circuito elétrico do Exemplo 13

Resolução:

Cálculo da resposta ao impulso:

Lei dos nós

$$i = i_R + i_C$$

$$i = i_R + C\frac{dV}{dt}$$

Em t = 0, i = δ(t) e $i_R = 0$

$$\frac{1}{C}.\delta(t) = \frac{dV}{dt} \rightarrow \boxed{V(0) = V(0^+) = \frac{1}{C}}$$

$$0 = \frac{V}{R} + C\frac{dV}{dt} \rightarrow \text{Resolução Eq. Diferencial} \rightarrow V(t) = \frac{1}{C}.e^{-t/RC}$$

$$i(t) = h(t) = C\,\frac{dV}{dt} = -\frac{1}{RC}e^{-t/RC}$$

$$\text{Para } R = 40\ k\Omega \text{ e } C = 300\ \mu F \rightarrow \boxed{h(t) = -\frac{1}{12}e^{-t/12}\ A}$$

Cálculo da convolução analiticamente:

$$i_C(t) = y(t) = \int_{-\infty}^{t} x(\lambda)h(t-\lambda)d\lambda = \int_{0}^{t} e^{-\lambda}\left(-\frac{1}{12}e^{-(t-\lambda)/12}\right)d\lambda$$

$$= -\frac{e^{-(t-\lambda)/12}}{12}\cdot\int_{0}^{t} e^{-11\lambda)/12}d\lambda = \frac{e^{-t/12} - e^{-t}}{11}$$

Cálculo da convolução através do MATLAB:

```
>> n = [0:0.08:81.84];     %amostragem
>> x = exp(-n);            %definição da entrada
>> h = -exp(-n/12)/12;     %definição da saída
>> fftx = fft(x);          %cálculo da fft
>> ffth = fft(h);
>> ffty = fftx.*ffth;      %multiplicação
>> y = ifft(ffty);         %inversa
>> plot(n,-abs(y)*0.08)
>> title('Convolução');
>> xlabel('t(s)');
>> ylabel('i(A)');
```

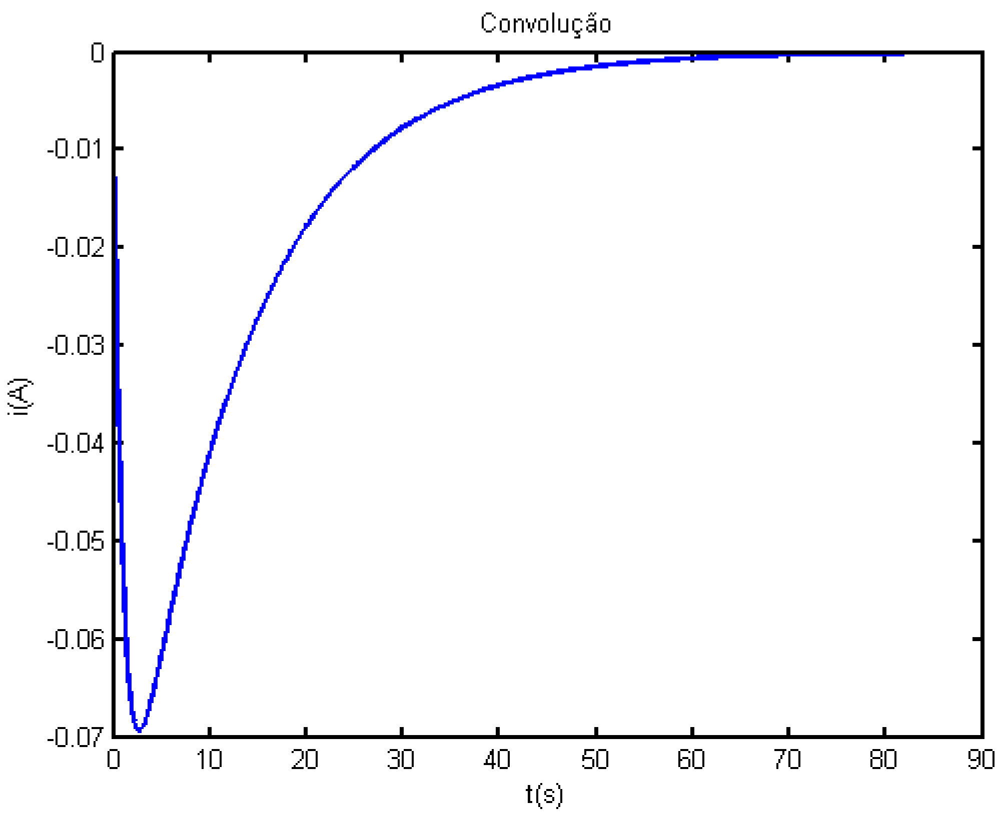


Figura 48 - Gráfico da resposta ao impulso do exemplo 13.

Exercício 25 - Calcule a convolução das formas de onda:

$x(t) = e^{-5t}$ e $h(t) = \cos(2{,}5t) - t$

13.6. Filtros Digitais

O MATLAB possui inúmeras funções que permitem ao usuário descobrir a função transferência de diferentes tipos de filtros digitais:

A função de transferência digital é definida por H(z) onde forma geral a função de transferência H(z) é a seguinte:

$$H(z) = \frac{B(z)}{A(z)}$$

$$H(z) = -\frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_n z^{-n}}{a_0 + a_1 z^{1} + a_2 z^{2} + \cdots + a_n z^{-n}}$$

- ***butter***

Definição: Determina os coeficientes de um filtro *Butterworth*. Esse filtro pode ser passa baixa, passa alta, passa banda, rejeita banda.

Sintaxe:

[B,A] = *butter*(N, Wn, 'tipo') → Determina os coeficientes da função transferência dada a freqüência de corte e o tipo de filtro. Caso nada seja posto em 'tipo', o MATLAB interpreta filtro passa baixa como padrão.

- ***freqz***

Definição: Calcula os valores de uma função complexa H(z)

Sintaxe:

freqz(B,A,n) → Utiliza 3 argumentos de entrada. O primeiro é um vetor contendo os coeficientes do polinômio B(z) da Equação (1). O segundo é um vetor contendo os coeficientes do polinômio A(z). O terceiro é para especificar o número de valores de freqüências normalizadas que se quer no intervalo de 0 a π.

Exemplo 14 -

a. Gerar um sinal com duas senóides de freqüências 5 e 80 Hz, com fs=200Hz.

b. Projetar um filtro para fs=200 Hz. Usar filtro de 2ª ordem, *Butterworth*.

c. Filtrar o sinal.

d. Plotar a resposta em freqüência.

```
%Exemplo de filtros
>> fs=200;                           %Freqüência de amostragem
>> t=0:1/fs:1;                       %Tempo de amostragem
>> T=1/fs;
>> x=sin(2*pi*5*t)+sin(2*pi*80*t);  %sinal de entrada
>> figure(4)
>> plot(t,x)
>> title('Sinal de Entrada')
>> xlabel('tempo (s)')
>> ylabel('amplitude')
>> [B,A]=butter(2,20/(fs/2));       %Determinar os coeficientes
>> B                                 %Mostrar coeficientes B
>> A                                 %Mostrar coeficientes A
%Plotagem da resposta em freqüência
>> h1 = freqz(B,A,100);
>> figure(1)
>> plot(abs(h1))
>> grid
>> title('Resposta em freqüência')
>> xlabel('freqüênca (Hz)')
>> ylabel('amplitude')
% Filtragem
>> figure(2)
>> y = filter(B,A,x);
>> plot(t,y,'k-')
>> title('Sinal de Entrada')
>> xlabel('tempo (s)')
>> ylabel('amplitude')
```

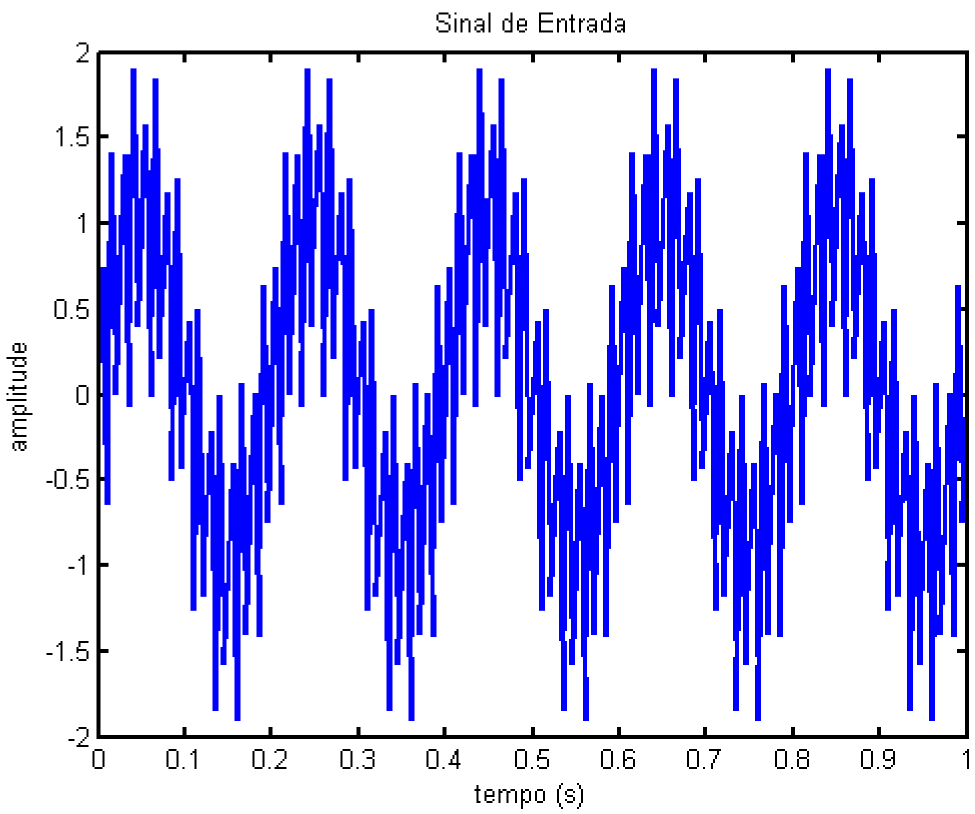


Figura 49 - Sinal de entrada não filtrado

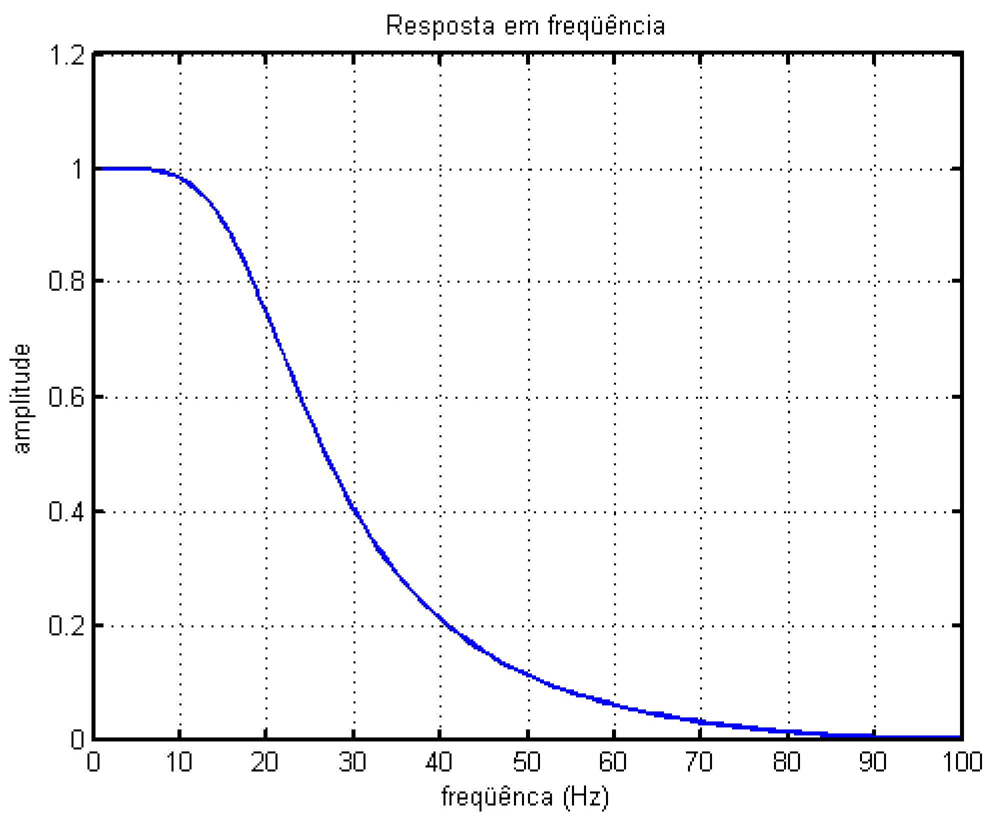


Figura 50 - Resposta em frequência

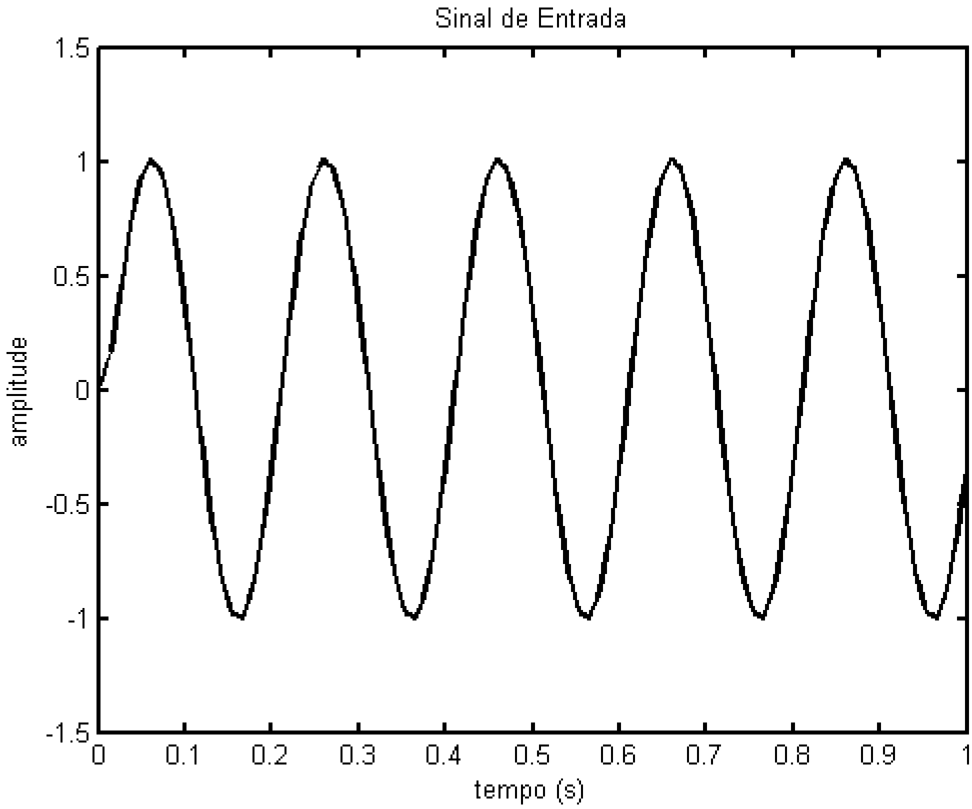


Figura 51 - Sinal de entrada filtrado

Exercício 26 - Projete um filtro passa-alta de *Butterworth* de ordem 6, com freqüência de corte de 10 Hz. Use fs=400 Hz. Sinais a serem filtrados: senóides de 1 e 20 Hz. Use as funções *butter*, *filter* e *freqz*.

14. CONTROLE

O MATLAB também é muito utilizado na área de Controle de Sistemas Discretos e Dinâmicos. A seguir estão as funções mais utilizadas nesse estudo:

- ***tf***

Definição: cria funções de transferência de tempo contínuo ou discreto no domínio da frequência

Sintaxe:

tf(num,den) → *Num* e *den* são vetores com os coeficientes do numerador e do denominador, respectivamente, da função de transferência.

```
>> num = [2 2];
>> den = [1 3];
>> tf(num,den)
```

- ***ss***

Definição: cria um modelo de equações de estados.

Sintaxe:

ss(A,B,C,D) → *A, B, C,e D* são as matrizes de estado da função de transferência.

```
>> A = [0, 3, -2; 1, 0, 0; 0, 1, 0];
>> B = [1; 0; 0];
>> C = [0, 1, 3];
>> D = [0];
>> G = ss(A,B,C,D)
```

- ***zpk***

Definição: cria funções de transferência de tempo contínuo na forma de zeros, pólos e ganhos

Sintaxe:

zpk(z,p,k) → z, p são vetores com os zeros e os pólos, respectivamente, e k o ganho da função de transferência.

```
>> z = -1;
>> p = -3;
>> k = 2;
>> zpk(z,p,k)
```

- ***tf2ss***

Definição: converte os coeficientes de uma função de transferência em zeros, pólos e ganho

Sintaxe:

[A,B,C,D] = *tf2ss(num,den)*

```
>> num = [1 3];
>> den = [1 0 -3 2];
>> [A,B,C,D] = tf2ss(num,den);
>> G = ss(A,B,C,D)
```

- ***ss2tf***

Definição: converte os coeficientes de uma função de transferência em zeros, pólos e ganho

Sintaxe:

[num,den] = *ss2tf(A,B,C,D)*

```
>> A = [0, 3, -2; 1, 0, 0; 0, 1, 0];
>> B = [1; 0; 0];
>> C = [0, 1, 3];
>> D = [0];
>> [num,den] = ss2tf(A,B,C,D);
>> G = tf(num,den)
```

- ***zp2ss***

Definição: converte os coeficientes de uma função de transferência em zeros, pólos e ganho

Sintaxe:

[A,B,C,D] = *zp2ss(z,p,k)*

```
>> z = -3;
>> p = [-2 1 1];
>> k = 1;
>> [A,B,C,D] = zp2ss(z,p,k);
>> G = ss(A,B,C,D)
```

- ***ss2zp***

Definição: converte os coeficientes de uma função de transferência em zeros, pólos e ganho

Sintaxe:

[z,p,k] = *ss2zp(A,B,C,D)*

```
>> A = [0, 3, -2; 1, 0, 0; 0, 1, 0];
>> B = [1; 0; 0];
>> C = [0, 1, 3];
>> D = [0];
>> [z,p,k] = ss2zp(A,B,C,D);
>> G = zpk(z,p,k)
```

- ***tf2zp***

Definição: converte os coeficientes de uma função de transferência em zeros, pólos e ganho

Sintaxe:

[z,p,k] = *tf2zp(num,den)*

```
>> num = [3 9];
>> den = [1 2];
>> [z,p,k] = tf2zp(num,den)
```

- ***zp2tf***

Definição: converte os zeros, pólos e ganho nos coeficientes de uma função de transferência

Sintaxe:

[num,den] = *zp2tf(z,p,k)*

```
>> z = -3;
>> p = -5;
>> k = 5;
>> [num,den] = zp2tf(z,p,k)
```

- ***tfdata***

Definição: retorna os coeficientes do numerador e do denominados de uma função de transferência em SS ou ZPK

Sintaxe:

[num,den] = *tfdata(sys)*

```
>> z = -1;
>> p = -3;
>> k = 2;
>> sys =zpk(z,p,k)
>> [num,den] = tfdata(sys)
```

- ***zpkdata***

Definição: retorna os zeros, os pólos e o ganho de uma função de transferência em SS ou ZPK

Sintaxe:

[z,p,k] = *zpkdata(sys)*

```
>> num = [3 9];
>> den = [1 2];
>> sys = tf(num,den);
>> [z,p,k] = zpkdata(sys)
```

- ***c2d***

Definição: converte uma FT em tempo contínuo (s) para tempo discreto (z).

Sintaxe:

c2d(sys, Ts) → Ts corresponde ao tempo de amostragem da FT em tempo discreto.

```
>> num = [3 9];
>> den = [1 2];
>> sys = tf(num,den);
>> Ts = 0.1;
>> c2d(sys,Ts)
```

- ***d2c***

Definição: converte uma FT em tempo discreto (z) para tempo contínuo (s).

Sintaxe:

d2c(sys, metodo) → Onde método corresponde ao método de conversão de contínuo para discreto.

```
>> num = [1 -2];
>> den = [1 -1];
>> Ts = 0.1;
>> sys = tf(num,den,Ts);
>> d2c(sys,'tustin')
```

- ***impulse***

Definição: plota a resposta ao impulso de uma função de transferência.

Sintaxe:

step(sys)

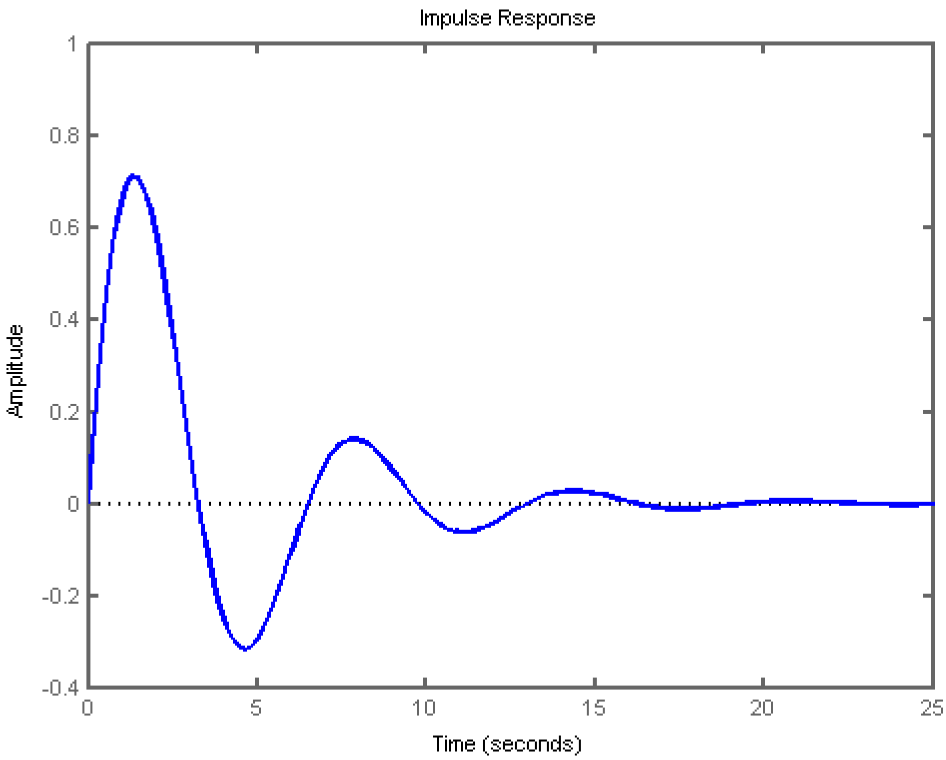


Figura 52 - Resposta ao impulso

- ***step***

Definição: plota a resposta ao degrau ou à rampa de uma função de transferência.

Sintaxe:

step(sys)

step(sys,t) → t representa o sinal rampa de entrada

```
>> num = [1 2];
>> den = [1 1];
>> G = tf(num,den);
>> step(G)
>> figure
>> num = [1];
>> den = [1 0.5 1 0];
>> t = 0:0.1:10;
>> G = tf(num,den);
>> y = step(G,t)
>> plot(t,y,t,t)
```

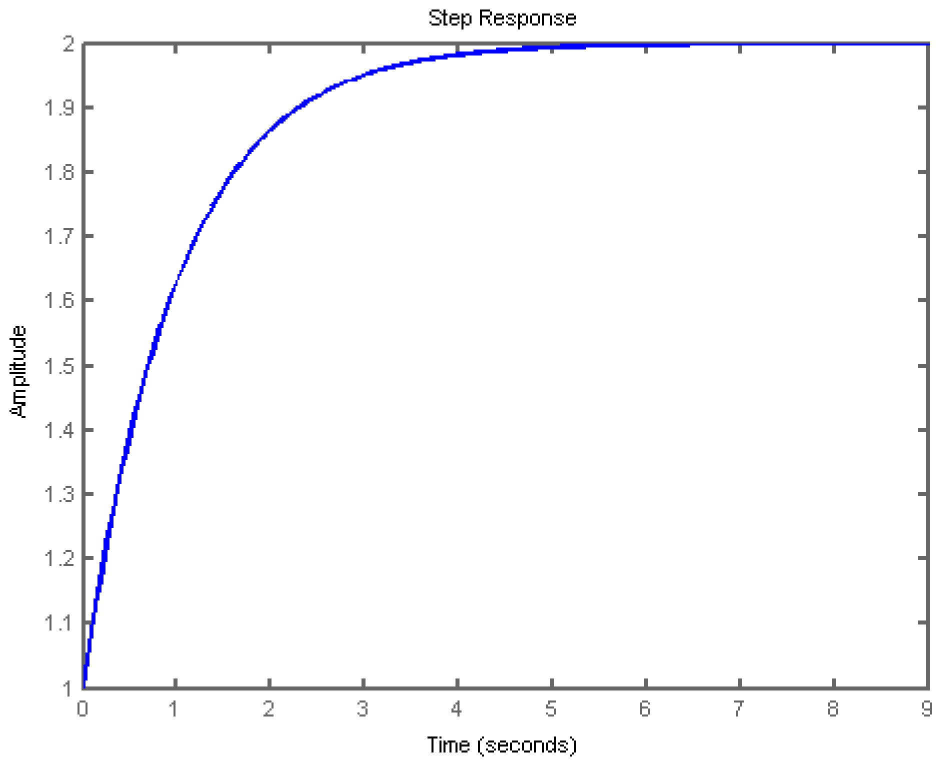


Figura 53 - Resposta ao degrau

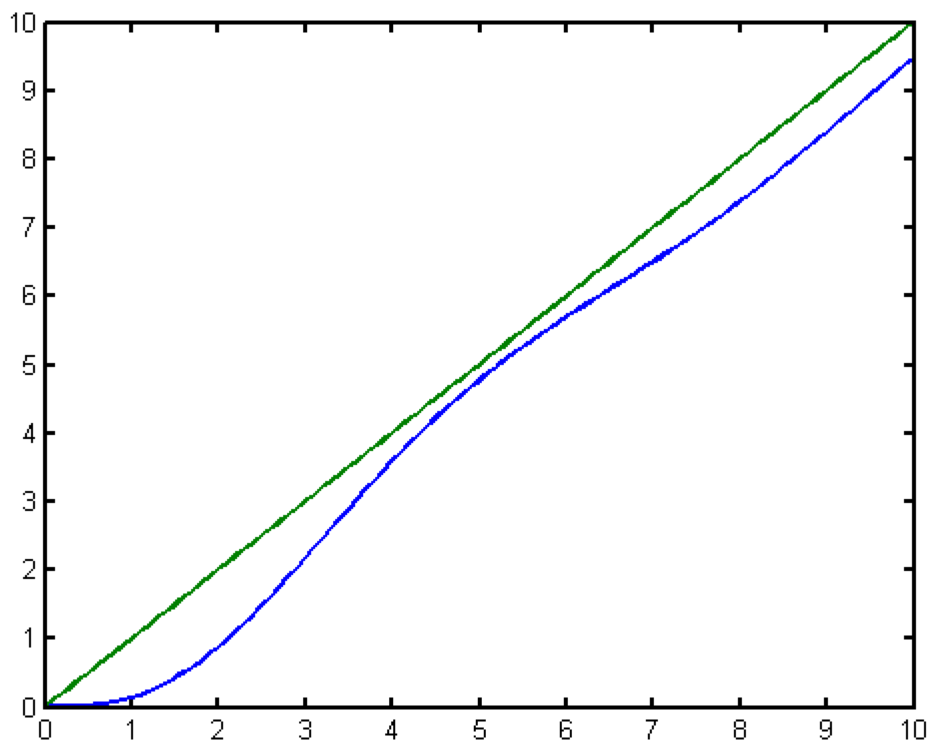


Figura 54 - Resposta à rampa

- ***series***

Definição: calcula a função de transferência equivalente de duas FTs em série. Equivalente a multiplicação das duas funções.

Sintaxe:

series(sys1,sys2)

```
>> G1 = tf(1,[1 1]);
>> G2 = tf(1,[0.5 1]);
>> G = series(G1,G2)
```

- ***parallel***

Definição: calcula a função de transferência equivalente de duas FTs em paralelo. Equivalente a soma das duas funções.

Sintaxe:

parallel(sys1,sys2)

```
>> G1 = tf(1,[1 1]);
>> G2 = tf(1,[0.5 1]);
>> G = parallel(G1,G2)
```

- ***feedback***

Definição: calcula a função de transferência equivalente da realimentação de uma funnção de transferência por outra.

Sintaxe:

feedback(sys1,sys2)

```
>> num = 10;
>> den = [0.5 1];
>> G1 = tf(num,den);
>> G = feedback(G1,1)
```

- ***lsim***

Definição: plota a resposta a um sinal de entrada genérico de uma função de transferência.

Sintaxe:

lsim(sys,u,t) → U representa o vetor com os valores de entrada e t o vetor com os valores de instante de tempo.

```
>> t = 0:0.1:5;
>> u = ones(1,length(t));
>> G = tf(1, [0.5 1]);
>> lsim(G,u,t)
```

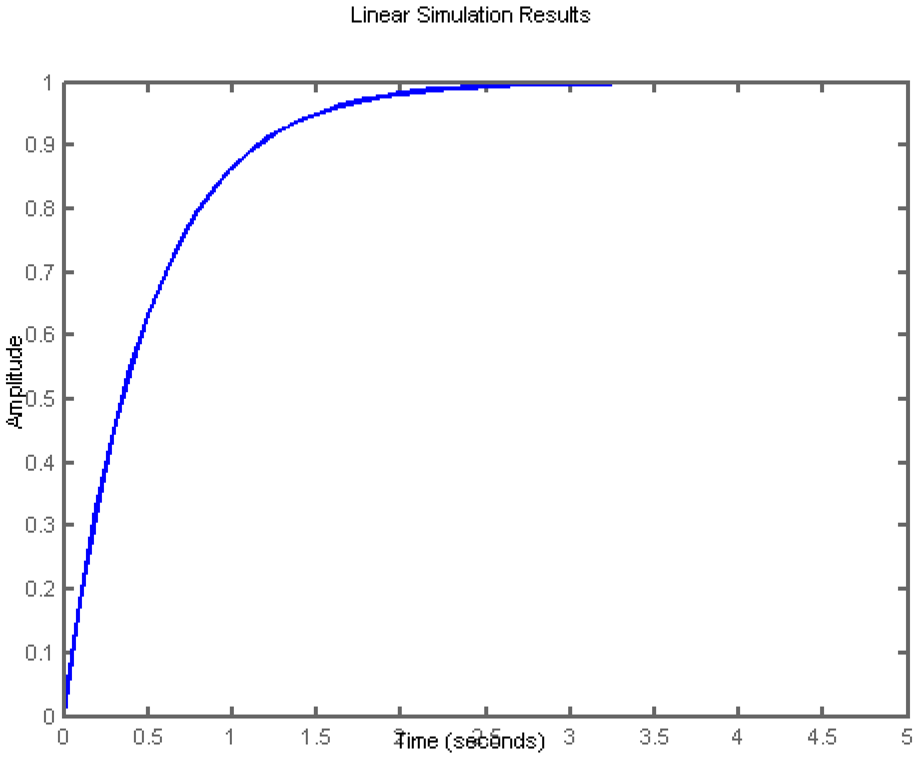


Figura 55- Resposta lsim de um sistema

- ***pzmap***

Definição: plota os pólos e zeros de uma função de transferência.

Sintaxe:

pzmap(sys)

```
>> num = [1 2];
>> den = [1 1 0];
>> G = tf(num,den);
>> pzmap(G)
```

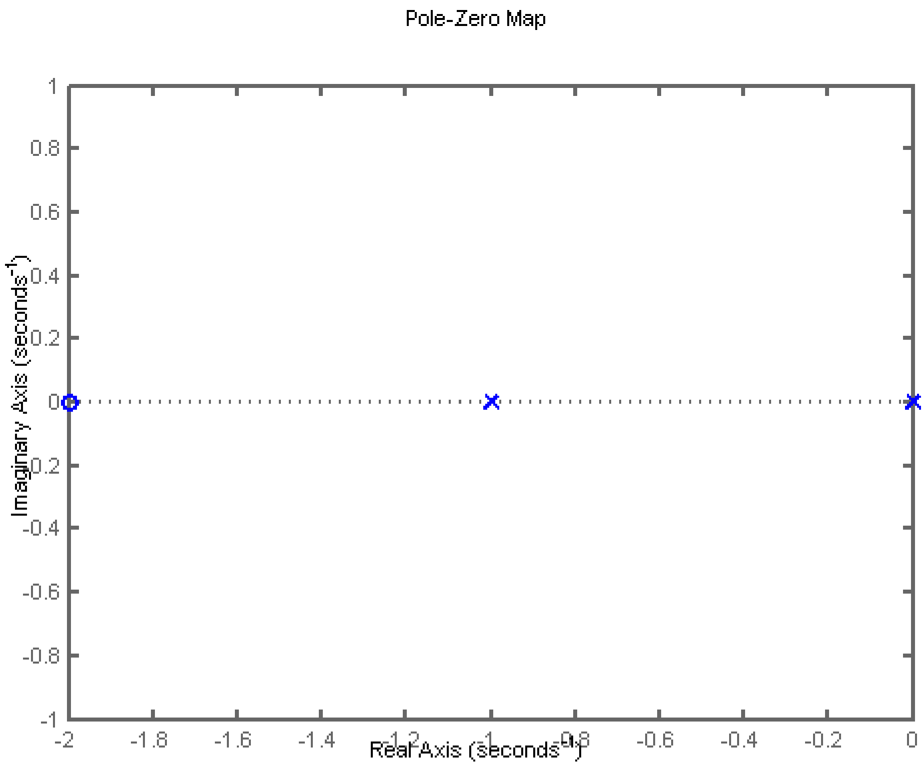


Figura 56 - Lozalização de pólos e zeros de uma função de transferência

- ***rlocus***

Definição: plota o lugar geométricos das raízes de uma função de transferência

Sintaxe:

rlocus(sys)

```
>> num = [1 2];
>> den = [1 1 0];
>> G = tf(num,den);
>> rlocus(G)
```

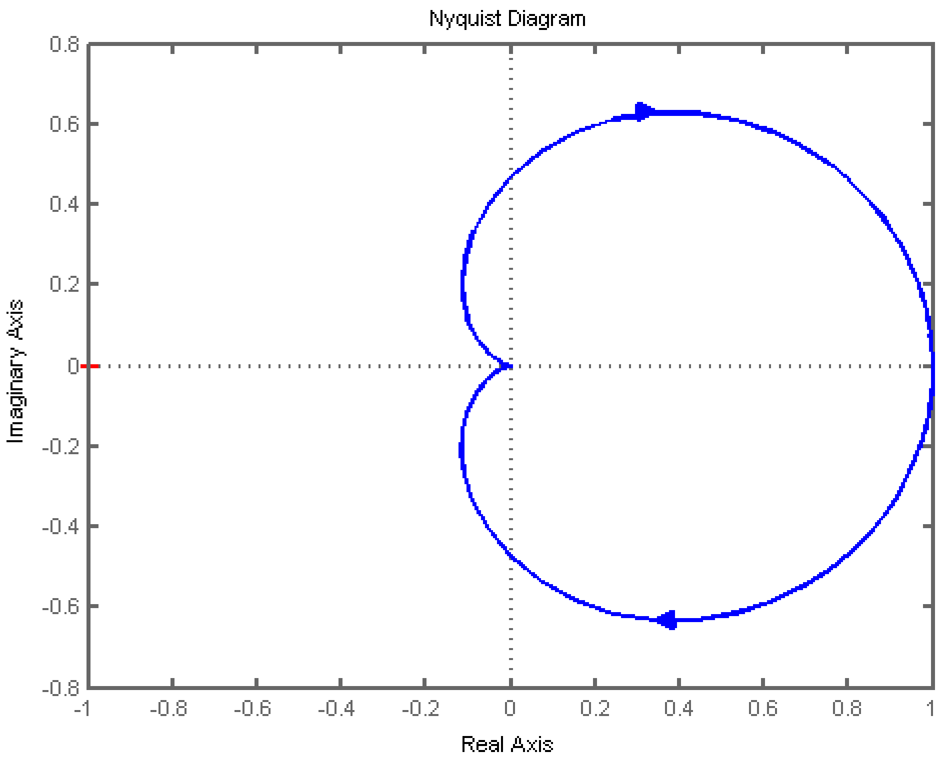


Figura 57 - Lugar geométricos das raízes de uma função detransferência

- ***nyquist***

Definição: plota o diagrama de *Nyquist* de uma função de transferência

Sintaxe:

nyquist(sys)

```
>> num = 1;
>> den = [0.5 1.5 1];
>> G = tf(num,den);
>> nyquist(G)
```

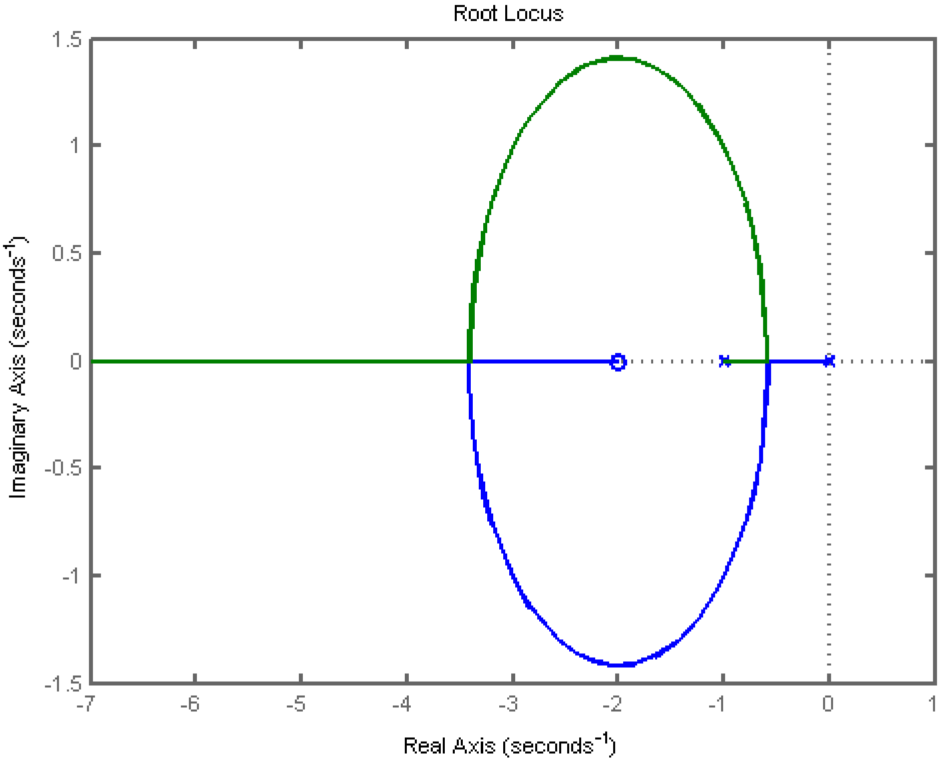


Figura 58 - Diagrama de Nyquist de uma função de transferência

- ***bode***

Definição: plota o diagrama de *bode* de uma função de transferência

Sintaxe:

bode(sys)

```
>> num = [1 5];
>> den = [1 1.5 1];
>> G = tf(num,den);
>> bode(G)
```

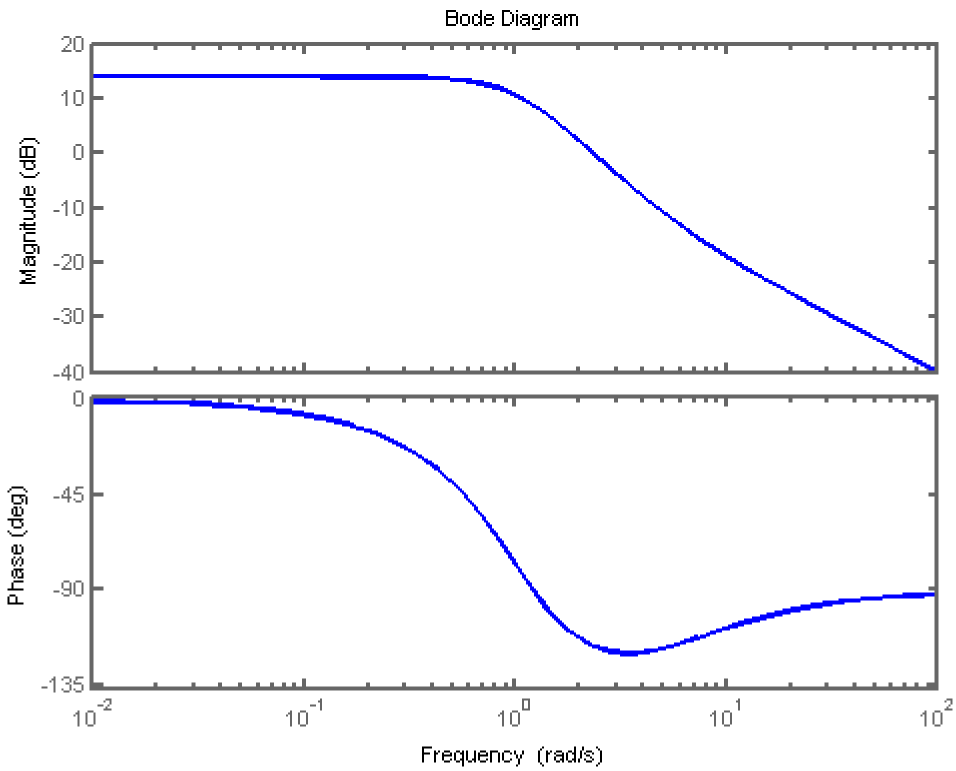


Figura 59 - Diagrama de Bode de uma função de transferência

- ***sigma***

Definição: plota o diagrama de *magnitude* de uma função de transferência

Sintaxe:

sigma(sys)

```
>> num = [1 5];
>> den = [1 1.5 1];
>> G = tf(num,den);
>> sigma(G)
```

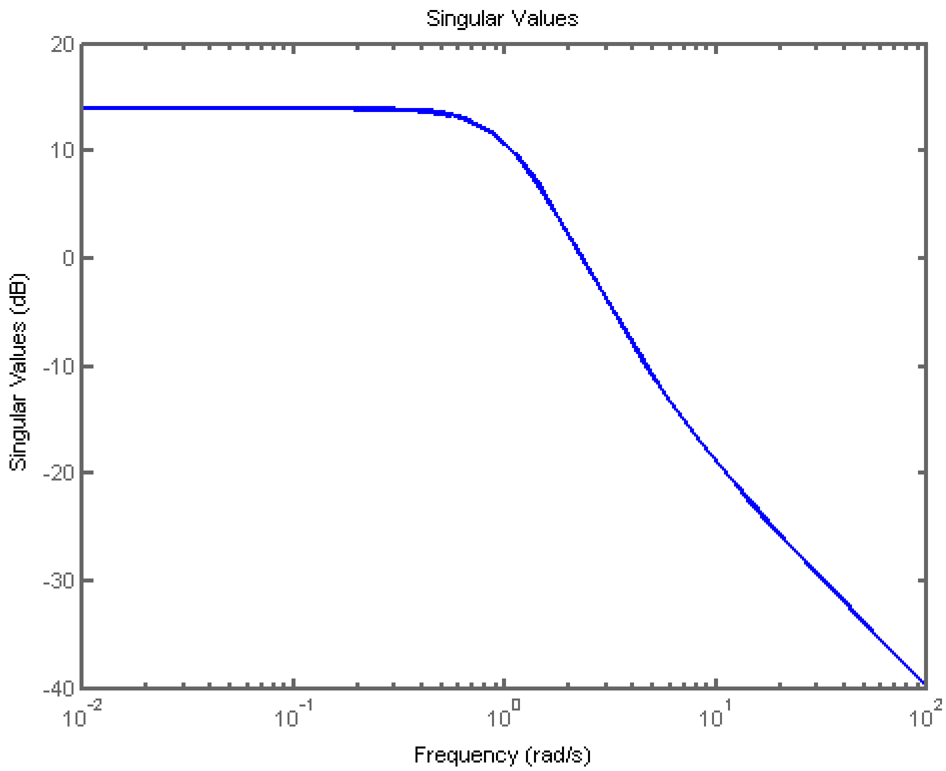


Figura 60 - Diagrama de magnitude de uma função de transferência

- ***sigma***

Definição: plota o diagrama de *bode* de uma função de transferência indicando as margens de ganho e de fase.

Sintaxe:

marginsys)

```
>> num = [1 5];
>> den = [1 1.5 1];
>> G = tf(num,den);
>> margin(G)
```

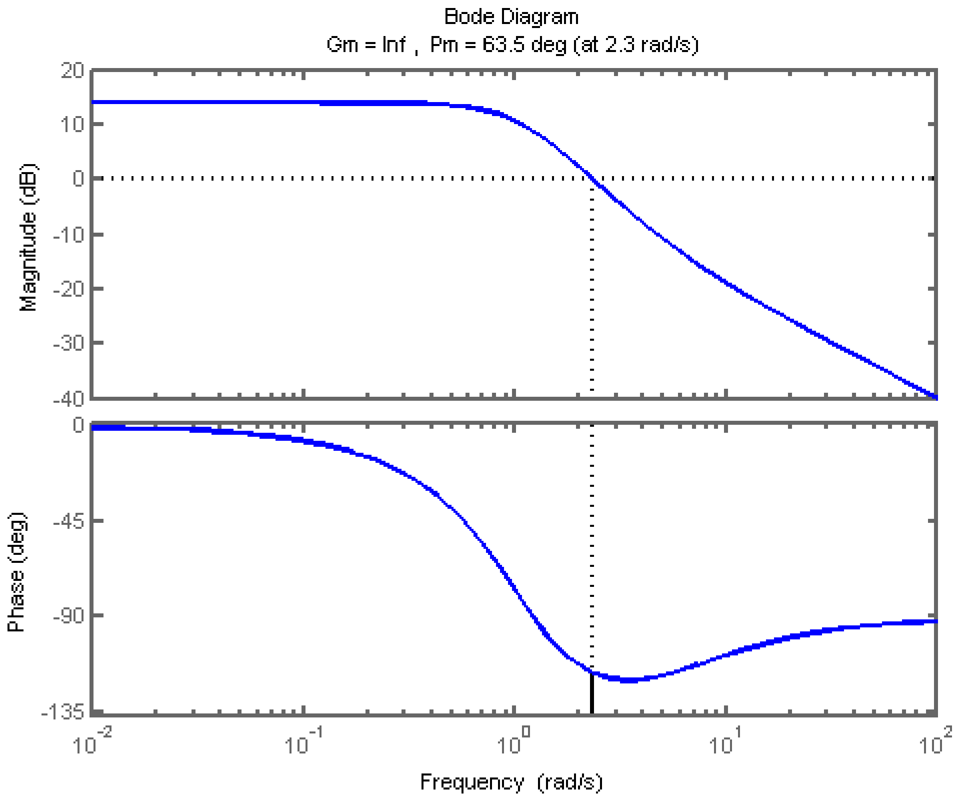


Figura 61 - Diagrama de bode de uma função de transferência

- ***stepinfo***

Definição: calcula as características da resposta ao degrau de uma função de transferência

Sintaxe:

stepinfo(sys)

```
>> num = 1;
>> den = [0.5 1.5 1];
>> G = tf(num,den);
>> stepinfo(G)
%ans =
%         RiseTime: 2.5901
```

```
%     SettlingTime: 4.6002
%      SettlingMin: 0.9023
%      SettlingMax: 0.9992
%        Overshoot: 0
%       Undershoot: 0
%             Peak: 0.9992
%         PeakTime: 7.7827
```

15. SIMULINK

O Simulink é um ambiente pertencente ao MATLAB que permite a simulação e a modelagem de sistemas dinâmicos e embarcados. É um ambiente gráfico e customizável que possui um conjunto de bibliotecas que facilitam a implementação e o teste de uma variedade de sistemas variantes no tempo.

Os modelos no Simulink são construídos através de diagramas de blocos, em operações do tipo “clique e arraste”, o que o torna uma interface bastante amigável. Ele permite a criação de qualquer máquina, artefato ou aparelho não existente no mundo real, através da modelagem matemática, e permite que o determinado sistema criado funcione virtualmente, através da resolução matemática do sistema criado.

15.1. Iniciando o Simulink

Para iniciar o Simulink, basta digitar o seguinte comando no *Workspace*:

```
>> simulink
```

Uma janela será aberta, Fig. 62, que é a biblioteca de blocos do Simulink:

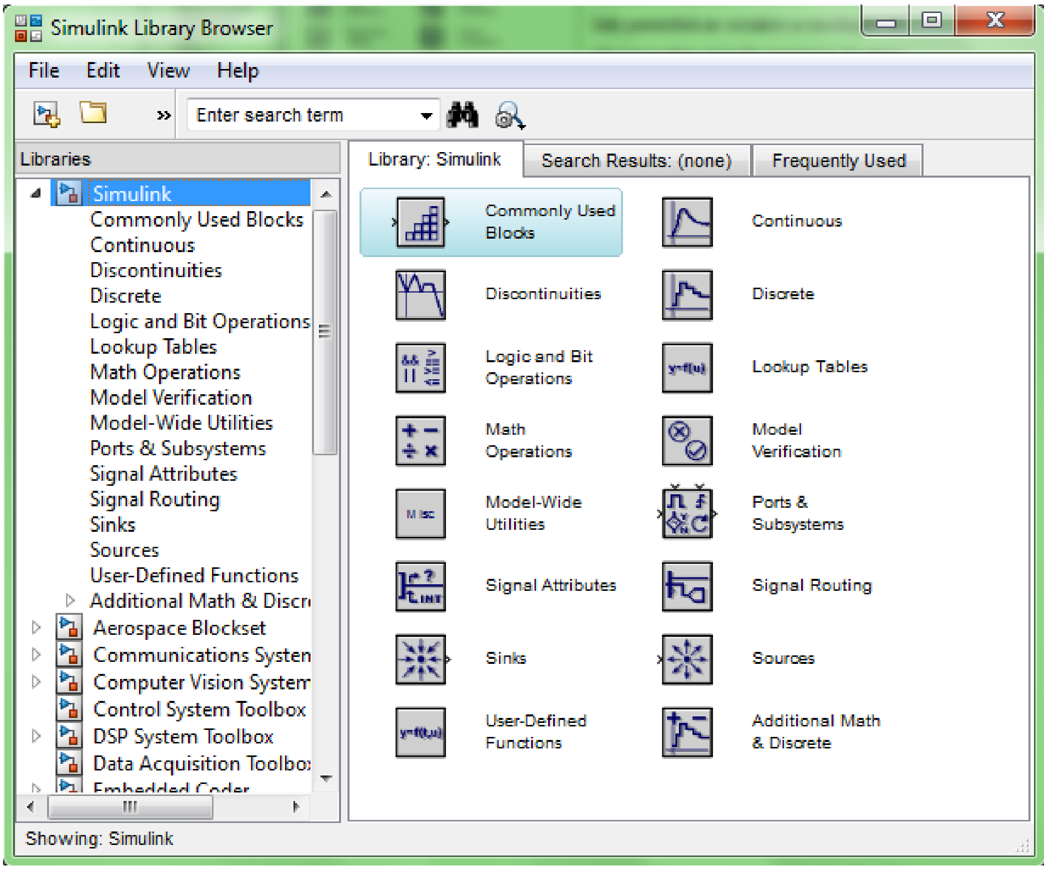


Figura 62 - Bibliotecas do Simulink

A janela é subdividida em três partes principais. No lado esquerdo estão presentes todas as bibliotecas disponíveis. Estão presentes bibliotecas de Sistema Aeroespaciais, Sistemas de Controle Dinâmicos, Sistemas Embarcado, Lógica *Fuzzy*, etc. No lado direito da tela estão disponíveis todos os blocos pertencentes à biblioteca selecionada. Na parte inferior da tela está presente a descrição de cada bloco selecionado.

Assim, quando se deseja saber o comportamento de um complexo Sistema Automotivo, da vibração nas asas de uma Aeronave durante o seu vôo, ou do efeito das futuras estimativas de oferta da moeda no Sistema Econômico, basta se dirigir ao MATLAB que o Simulink permitirá ao usuário a resolução do problema em sua própria casa, através do computador, não sendo necessárias grandes pesquisas de campo ou modernasa parelhagens em laboratórios.

15.2. Criando um modelo

Para a criação de um novo modelo é necessário abrir uma nova janela de modelo. Para isso basta abrir o menu File e escolher a opção *New Model*, Fig. 63.

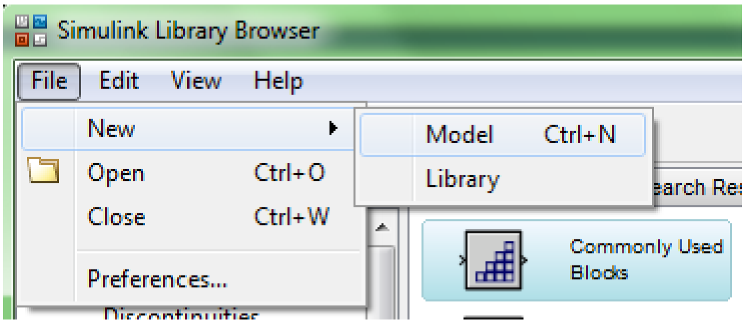


Figura 63 - Novo modelo do Simulink

Uma janela será então aberta. Essa é a janela de modelos do Simulink. Toda a montagem, modificação e testes dos sistemas são feitas nessa janela. O MATLAB salva esse arquivo de simulação com a extensão .mdl e, assim como no M-File, para que o sistema seja executado é necessário que ele esteja salvo. Como um primeiro exemplo, pode-se mostrar na tela uma onda senoidal. Para montar esse sistema são necessários dois dispositivos: primeiro algum dispositivo que seja capaz de mostrar a onda e também a própria onda.

Para adicionar uma onda senoidal seleciona-se a biblioteca *Source* na janela de bibliotecas do Simulink, Fig. 64. Quando selecionamos essa biblioteca, aparece no lado direito um conjunto de sinais que podem ser usados como entrada do sistema, tais como sinal de Rampa, sinal de Degrau, onda senoidal, etc. O bloco referente à onda senoidal se chama *Sine Wave*. Para adicionar o bloco à janela basta clicar com o botão direito do mouse e escolher a opção "Add to nome_da_janela".

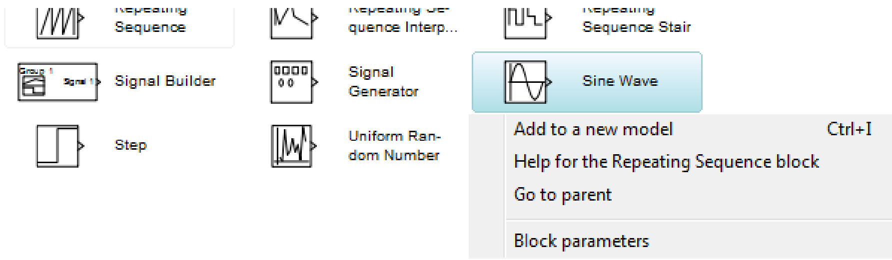


Figura 64 - Bloco *Sine Wave*

O bloco será adicionado na janela de modelos aberta. Tendo a forma de onda de entrada, basta ter o dispositivo capaz de mostrar a onda na tela. Esse dispositivo está presente na biblioteca *Sink*, o nome do bloco é *Scope*. Para adicionar esse bloco o procedimento é o mesmo. Depois dos blocos serem adicionados é necessário interligá-los. Para isso basta clicar na seta presente na lateral de um dos blocos, manter o botão pressionado e levar a linha tracejada até a seta presente no outro bloco. Se a cor da linha que liga os dois blocos ficar preta e o tracejado se manter contínuo, a operação foi realizada corretamente. O sistema está pronto, para executá-lo basta clicar no botão *Start Simulation*, na barra superior. Para controlar o tempo de simulação, basta digitar o mesmo na janela "*Simulation stop time*: 10.0", Fig. 65.

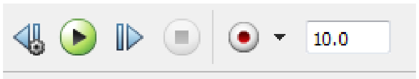


Figura 65 - Botões de simulação

O sistema foi simulado. Para visualizar a onda, dá-se um duplo clique no *Scope*. Surgirá na tela a forma de onda de *Sine Wave*. O resultado final pode ser visto na Fig. 66:

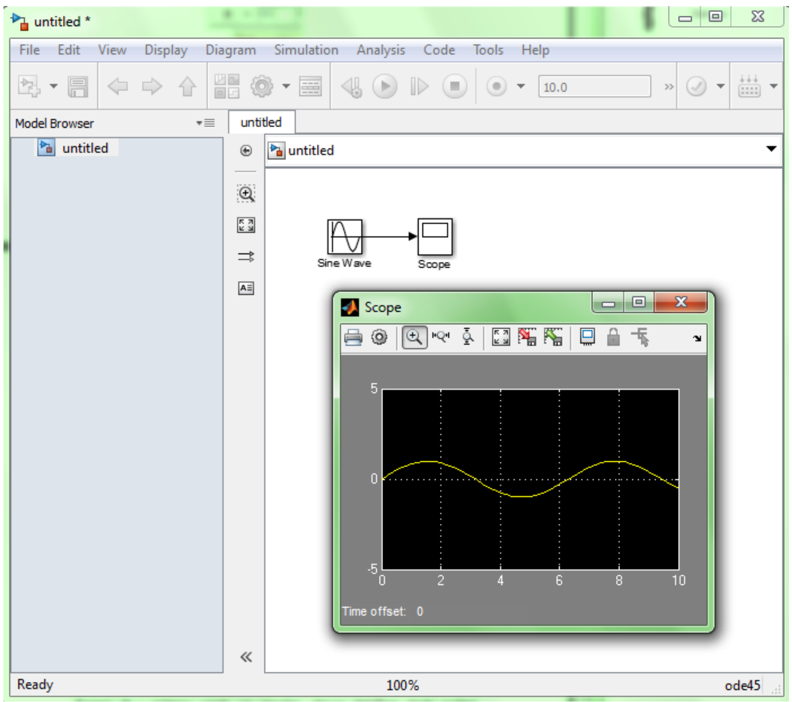


Figura 66 - Sinal visto pelo *Scope*

Apesar de o sistema criado ser simples, alguns detalhes ainda podem ser explorados. Ao se dar duplo-clique em *Sine Wave* veremos na tela algumas características da forma de onda gerada, Fig. 67. Aparecerá na tela uma janela onde podem ser modificados alguns parâmetros do sinal, tais como amplitude, freqüência, deslocamento vertical, ângulo de fase e domínio. Na parte superior da janela também pode ser observada a equação que descreve a onda.

São operações como essa de clique-e-arraste, de execução e de adição e interligação de blocos que governam o Simulink. Não é necessário um extenso programa com diversas diretivas e várias funções matemáticas. Tudo já está embutido nos blocos do Simulink. Esta simplicidade também pode ser mostrada no seguinte exemplo, onde é calculada a derivada do sinal senoidal utilizado anteriormente. Para isso pode-se utilizar o bloco *Derivative*, presente na biblioteca *Continuous*, Fig. 67:

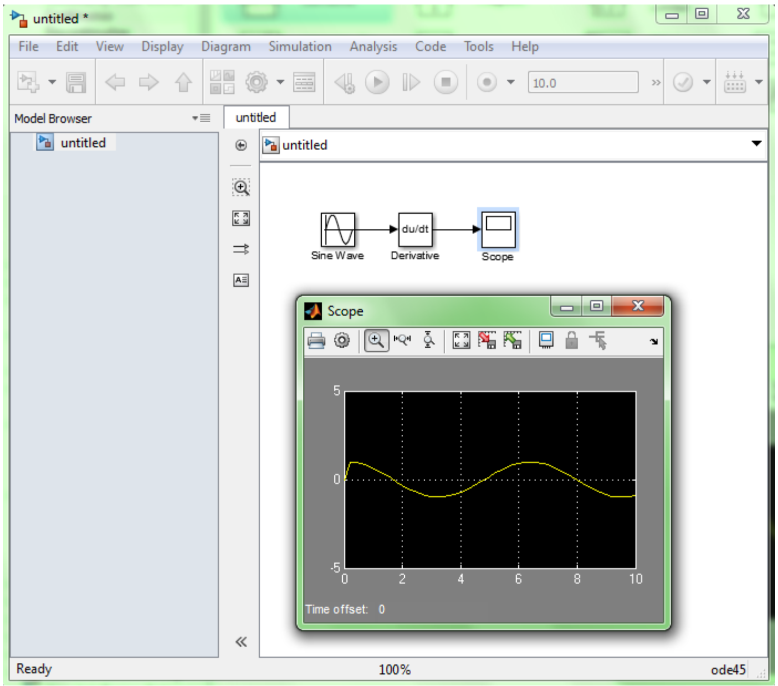


Figura 67 - Derivada do sinal vista pelo *Scope*

15.3. Aspectos sobre a solução dos sistemas

É evidente a presença de métodos numéricos na resolução dos sistemas no MATLAB. Observando as configurações de simulação, é notável o uso dos métodos. As configurações de simulação estão presentes no menu *Simulation*, na opção *Configuration Parameters*, Fig. 68.

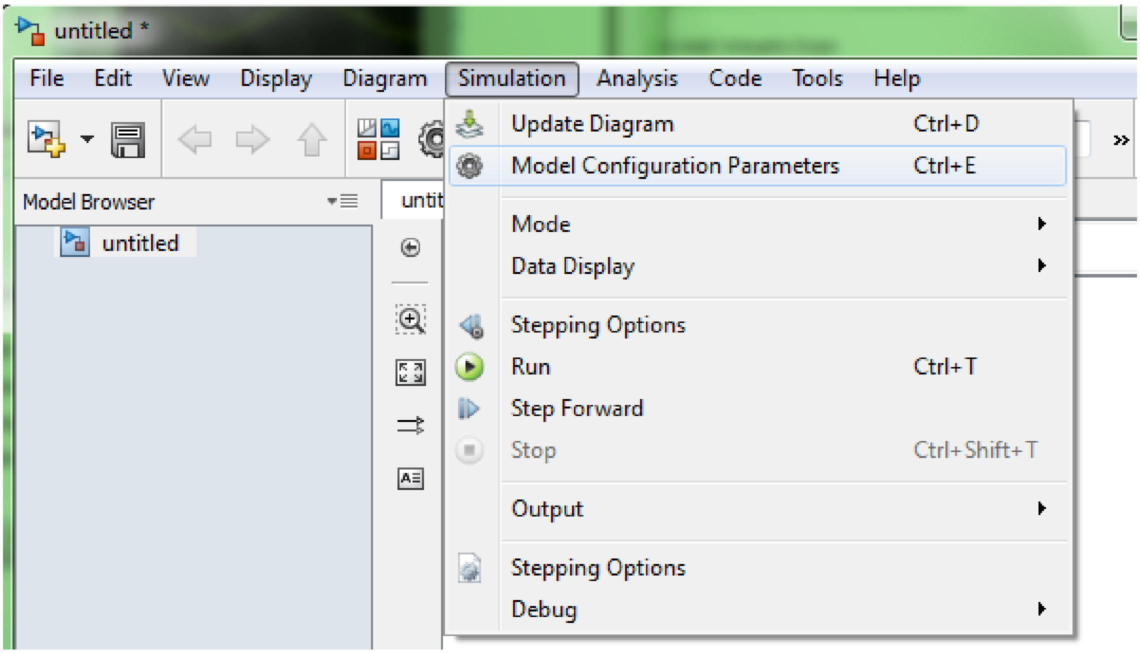


Figura 68 - *Configuration Parameters*

Um das modificações importantes que podem ser feitas é a do próprio método de resolução das equações diferencias. O MATLAB oferece algumas opções de Métodos:

ODE45 - Excelente método de propósito geral de passo simples. É baseado nos métodos de *Dormand-Prince* e de *Runge-Kutta* para Quarta/Quinta ordem. ODE45 é o método padrão do Simulink.

ODE23 - Usa os métodos *Bogacki-Shampine* e *Runge-Kutta* de Segunda ou Terceira ordem. Pode ser uma melhor opção que o ODE45. Geralmente requer um passo menor do que o ODE45 para atingir a mesma precisão.

ODE113 - Utiliza o método de ordem variável de *Adams-Bashforth-Moulton*. Já que ODE113 utiliza as soluções de pontos em tempos anteriores para se determinar a solução do tempo corrente, deve então produzir a mesma precisão dos métodos ODE45 ou ODE23 com menor número de cálculos e com isto tendo uma melhor performace. Não é apropriado para sistemas com descontinuidades.

ODE15S - Sistema de ordem variável de multi passos para sistemas inflexíveis. É baseado em pesquisas recentes que utilizam fórmulas numéricas de diferença. Se a simulação executar lentamente utilizando ODE45, tente ODE15S.

ODE23S - Ordem fixa de passo simples para sistemas inflexíveis. Devido ao fato de ser um método de passo simples, em muitas das vezes é mais rápido do que ODE15S. Se um sistema parecer inflexível é uma boa idéia tentar ambos os métodos para este tipo de sistema para se determinar qual dos dois tem melhor performace.

ODE23t - Implementação do método trapezoidal utilizando uma interpolação livre. Usar quando se quer solução sem um amortecimento.

ODE23TB - Implementação do TR-BDF2, um método de *Runge-Kutta* com primeiro estágio com uma regra trapezoidal e um Segundo estágio utilizando diferenciação retrógrada de ordem 2. Pode ser mais eficiente que o ODE45.

15.4. Modelagem de Sistemas

Tornando os exemplos mais práticos, iremos modelar dois sistemas matemáticos.

- **Sistema Massa-Mola**

O primeiro sistema físico a ser modelado será o sistema massa-mola apresentado na Fig, 69, que possui um equacionamento simples baseado na Física Básica.

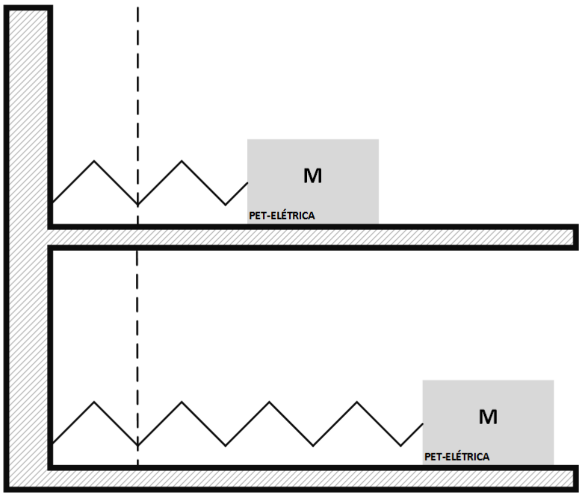


Figura 69 - Sistema Massa-Mola

Considerando uma aceleração ‘a’ para o bloco, teremos:

$v = \int a.dt$, onde v é a velocidade do bloco.

$x = \int v.dt$, onde x é o deslocamento do bloco.

A partir dessas equações um escopo geral do sistema já pode ser iniciado. Será utilizado o bloco *Integrator*, pertencente à biblioteca *Continuous*. Partindo da aceleração, Fig. 70, teremos:

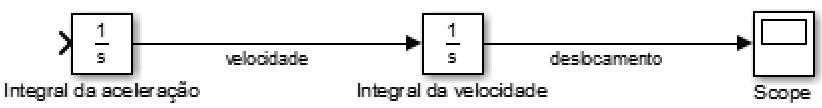


Figura 70 - Esquema no Simulink com bloco *Integrator*

Modelagem matemática do sistema:

F = - k.x , força restauradora exercida pela mola no bloco ao se aplicar uma deformação nesta.

$- kx = m \frac{d^2x}{dt^2}$

$a = \frac{d^2x}{dt^2} = -\frac{k}{m}x$, foi achada então uma relação entre a aceleração do sistema e o deslocamento. Essa relação é presente no sistema através de um ganho no deslocamento, o que resulta na aceleração. No utilizada a biblioteca *Math Operations*, dentro desta biblioteca se localiza o bloco Gain, Fig. 71:

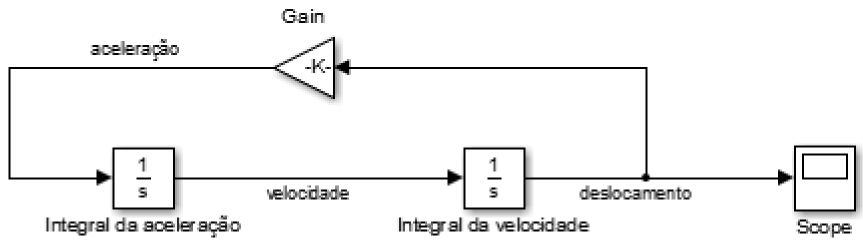


Figura 71 - Esquema do Simulink com bloco *Gain*

O bloco *Gain*, se não for modificado, terá o ganho 1. Para modificar o ganho deve-se clicar duas vezes no bloco. Aparecerá na tela a janela de configuração dos parâmetros do bloco, onde o ganho pode ser modificado, Fig. 72:

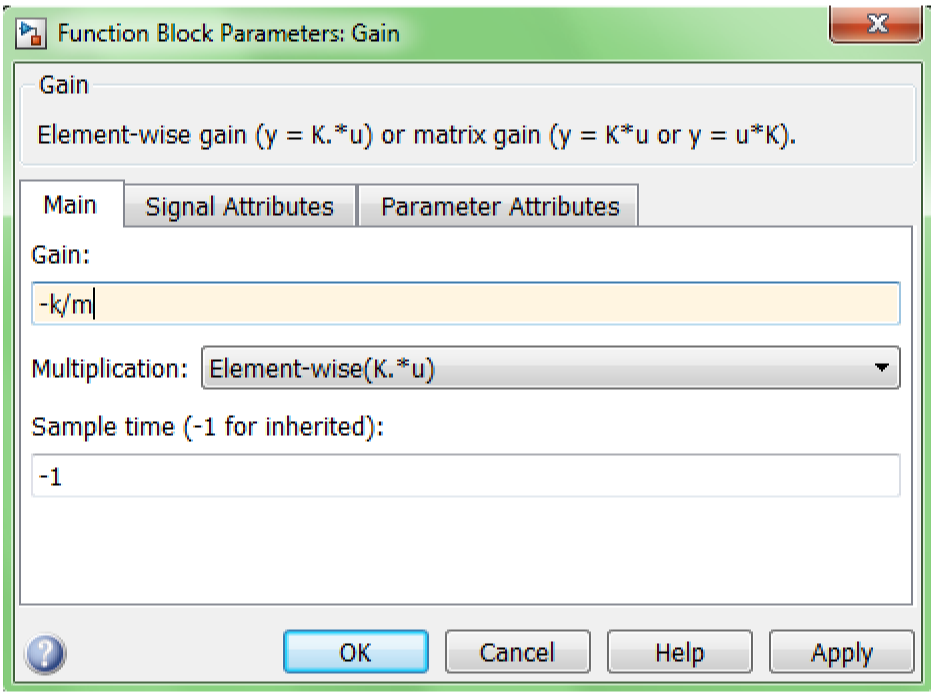


Figura 72 - Configuração dos parâmetros do bloco *Gain*

Como foi colocado um valor literal no ganho, quando a simulação for executada as variáveis ‘k’ e ‘m’ devem estar presentes no *workspace*. Antes de realizar a simulação pode-se definir as variáveis no *command window*:

```
>> k=500;

>> m=20;
```

O sistema está então pronto para ser simulado. É importante perceber que se a simulação for realizada, o valor do deslocamento será nulo. Isso acontece porque nenhum deslocamento inicial foi aplicado ao sistema. Para aplicar um deslocamento inicial pode-se modificar a condição inicial da integral da velocidade, pois:

$$x = \int_0^t v.dt + x_0$$

Quando a condição inicial é modificada para um valor maior que zero, estamos modificando o valor do deslocamento da mola no instante inicial do sistema, assim o que aconteceu foi que aplicamos uma entrada no sistema. Para modificar a condição inicial da integral da velocidade, é necessário se modificar os parâmetros da integral. O procedimento realizado é o mesmo que o do bloco *Gain*: dá-se um duplo-clique no bloco *Integrator*, o que fará aparecer na tela uma janela onde os parâmetros da integral possam ser modificados. Em *Initial condition*, pode-se colocar a condição inicial '2', Fig. 73:

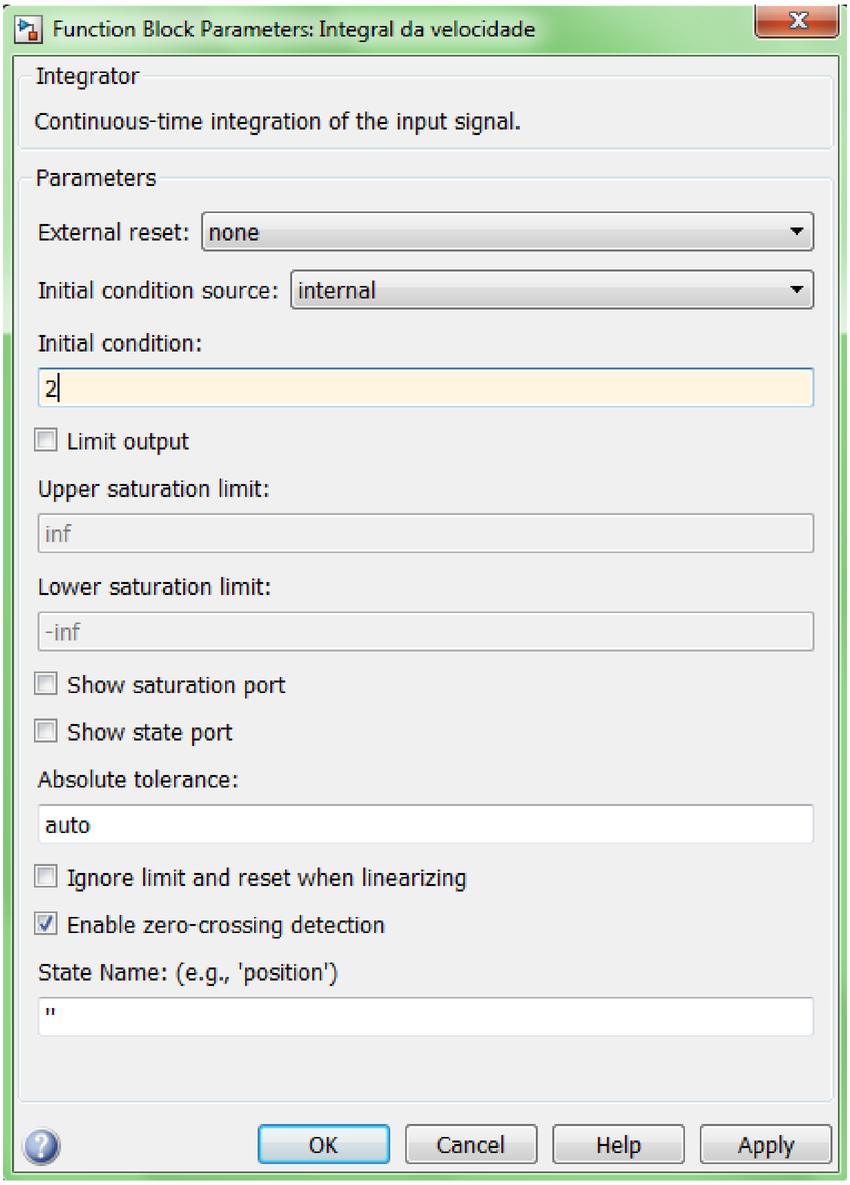


Figura 73 - Configuração dos parâmetros do bloco *Integrator*

O Sistema estará então pronto para ser simulado. Dando um duplo-clique em *Scope*, é possível observar o comportamento do deslocamento ao longo do tempo. Como já era esperado, o deslocamento se apresentou como um sinal oscilante em torno do ponto inicial, a amplitude. O modelo é diferente do real, pois não foram consideradas no equacionamento as equações dissipativas, como atrito com o ar e atrito com o chão por exemplo, Fig. 74.

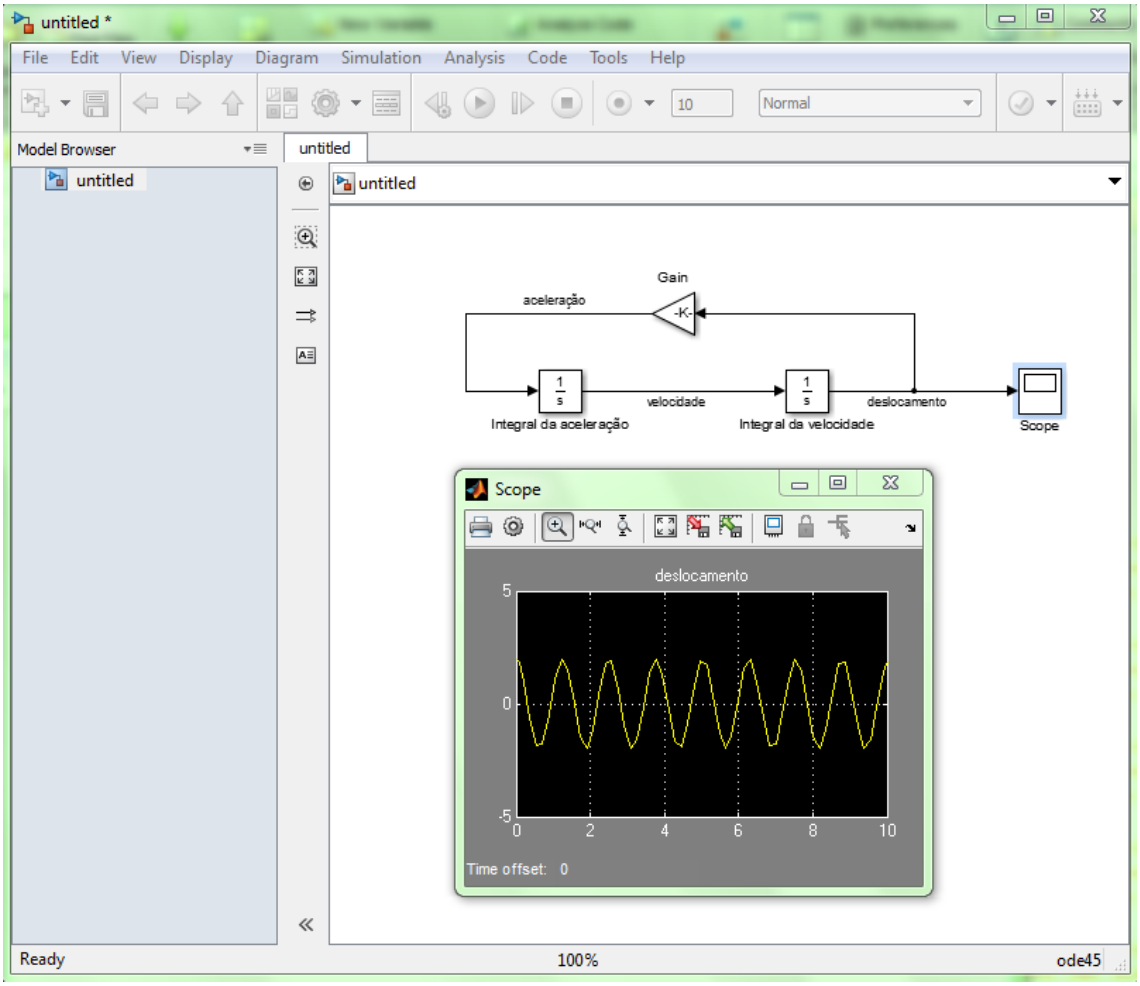


Figura 74 - Resposta do Modelo Massa-Mola

Para evidenciar o comportamento do sistema massa-mola clássico, pode ser traçado também a curva da energia potencial x energia cinética do bloco. Para isso tem-se que realizar outro equacionamento:

$E_P = \frac{1}{2} kx^2$, energia potencial do bloco

$E_C = \frac{1}{2} mv^2$, energia cinética do bloco

A energia potencial do sistema dependerá do deslocamento instantâneo, assim será necessária a saída do sistema, que é o próprio 'x'. Como se observa na equação, o primeiro passo será elevar o deslocamento ao quadrado. Uma opção para realizar essa operação é o bloco *Product*, presente em *Math Operations*. Esse bloco tem como configuração padrão, duas entradas e uma saída que é resultante da multiplicação das duas entradas. Assim ligamos a saída do sistema (deslocamento) simultaneamente às

duas entradas, resultando assim em ‘x^2’ na saída do bloco. O próximo passo será realizar um ganho de k/2 nesse sinal, o que resulta na energia potencial, Fig. 75:

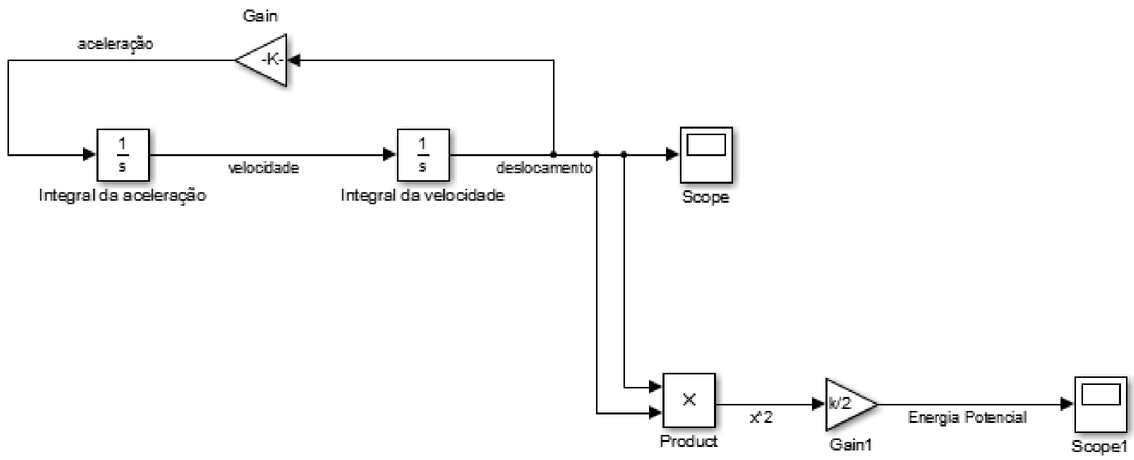


Figura 75 - Modelo Massa-Mola: Energia Potencial

O procedimento para se obter a energia cinética é análogo. Primeiro é preciso obter o sinal ‘v^2’ e depois é preciso dar um ganho de m/2 nesse sinal, resultando assim na energia cinética, Fig. 76. Para traçar os dois sinais em um só gráfico, utiliza-se apenas um *Scope*. Em conjunto utiliza-se também o bloco Mux, presente na biblioteca *Signal Routing*, Fig. 77:

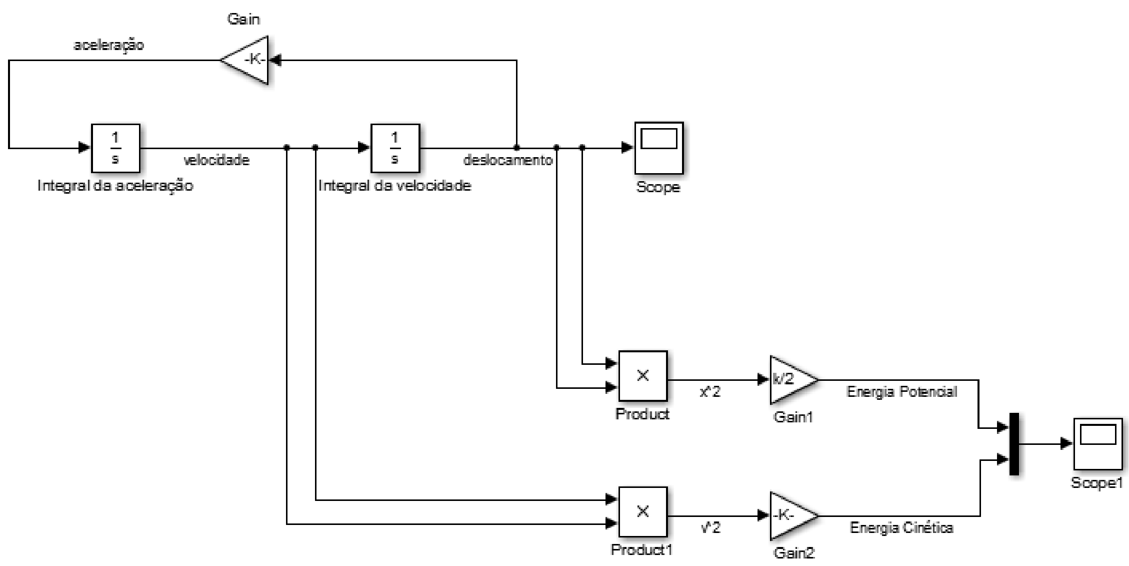


Figura 76 - Modelo Massa-Mola: Energia Potencial e Cinética

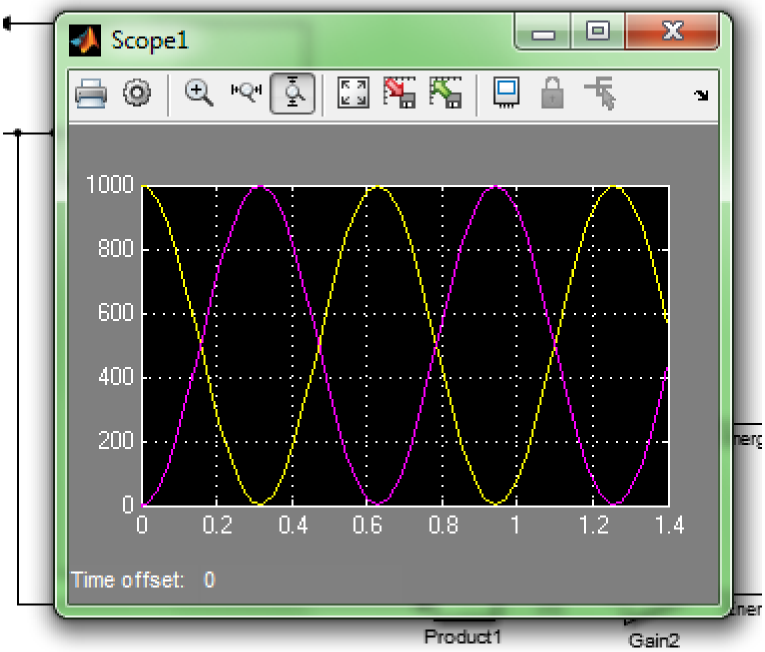


Figura 77 - Sinais de Energia Potencial e Cinética pelo Simulink

Através do gráfico pode-se observar uma característica típica dos sistemas conservativos, que é a complementaridade das duas energias ao longo do tempo. Pode-se observar que no início a energia potencial (em amarelo) é máxima, enquanto que a energia cinética (em roxo) é nula, devido a ausência de movimento. Situação oposta quando o bloco passa pela origem, em que a energia potencial é nula e a cinética é máxima.

- ## Amortização de Financiamento

O balanço de um empréstimo ao fim de um mês equivale aos juros incidindo sobre o balanço do início do mês menos o pagamento do mês. Uma equação para o processo é a seguinte:

balanço(k) = juros.balanço(k - 1) - pagamento(k)

Um sistema como esse também pode ser modelado principal diferença é que ele é um sistema discreto, enquanto massa-mola era um sistema contínuo.

Considerando um empréstimo feito por um petiano do curso de Engenharia Elétrica para a compra de um jatinho, no valor de R$ 5.000.000,00. Considere um regime amortizado segundo o modelo descrito acima, taxa de juros mensal fixa de 1%. O pagamento mensal realizado pelo petiano é de R$ 100.000,00. Calcule então a quantia que o petiano deverá ao banco ao final de 2 anos.

O primeiro passo para a modelagem do sistema é definir a sua saída. Como o petiano deseja saber quanto ele deverá ao banco no final dos dois anos, a solução desejada é o valor numérico de ‘balanço(24)’, já considerando a unidade temporal como o mês.

O balanço é dado por uma subtração, assim será necessário utilizar o bloco *Sum*, pertencente a biblioteca *Math Operations*. Quando se adiciona esse bloco ao modelo de sistema, pode-se observar o seguinte símbolo, Fig. 78:



Figura 78 - Bloco *sum* configurado com + +

O que se conclui a partir desse símbolo é que este bloco tem duas entradas e uma saída, que retorna a soma das entradas. Se for dado um duplo-clique, pode-se perceber na configuração dos parâmetros do bloco o parâmetro *List of Sign*, que conterá '|++'. Estes símbolos definem as configurações possíveis para o somador. Existem vária configurações, mas a requerida no problema do petiano é a diferença entre os termos da entrada. Trocando '|++' por '|-+', o bloco terá um novo aspecto, Fig. 79.



Figura 79 - Bloco *sum* configurado com - +

O bloco *Sum* será responsável por retornar o balanço equivalente do mês. Como observado na equação são entradas serão o pagamento mensal e o balanço do mês anterior. O pagamento, por ser de uma quantia fixa, pode ser representado por uma constante. O bloco *Constant*, presente em *Source* pode representar bem o pagamento. Ao se adicionar o bloco, dá-se duplo modo a trocar o valor da constante, de '1' para '100000', Fig. 80.



Figura 80 - Pagamento mensal

O outro termo do somado será o balanço do início do mês com os juros, ou seja, será necessário o balanço atrasado. Para se obter esse balanço é necessário realizar uma operação de atraso no balanço atual. O bloco que define a operação de atraso é denominado *Unit Delay*, está presente na biblioteca Discrete. Esse bloco atrasa o sinal de entrada em uma unidade. Como o sinal a ser atrasado é o próprio balanço, então a entrada desse bloco tem que ser colocada na saída do somador, Fig. 81.

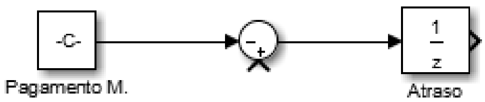


Figura 81 - Balanço atrasado

A outra entrada do somador será o balanço atrasado (saída de *Unit Delay*) multiplicado pelos juros, Fig. 82. Como o juro mensal é de 1%, será usado um bloco Gain com ganho de 1,01:

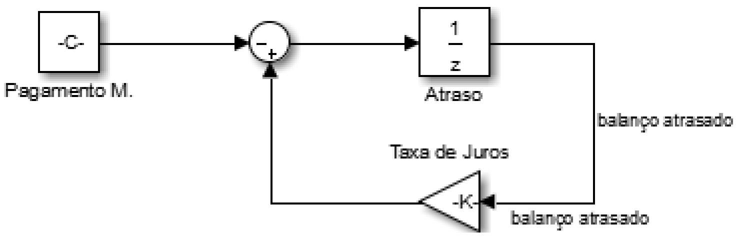


Figura 82 - Balanço mensal

Para uma melhor visualização dos cálculos feitos, será utilizado um *Scope* e um bloco chamado *Display*, presente na biblioteca *Sinks*. Esse bloco mostra o valor de uma determinada variável no instante em questão. A quantia inicial do financiamento é posta no sistema através do valor inicial do *Unit Delay*, que será nesse caso de R$ 5.000.000,00. O sistema está então pronto para ser simulado. Sempre é recomendável verificar os parâmetros de simulação em casos discretos. Primeiro verificar se a opção de solução escolhida é *discrete, para isso, na tela da Fig. 68,* clique em *Model Configuration Program* e modifique a opção *Solver* para *discrete* e o *Max step size* para 1. E segundo, verificar o tempo de simulação, presente em *Start Time* e em *Stop Time*. Como se quer o valor ao final de dois anos, deve-se escolher um *Stop Time* igual a 25 (devido ao atraso). O resultado será o seguinte, Fig. 84:

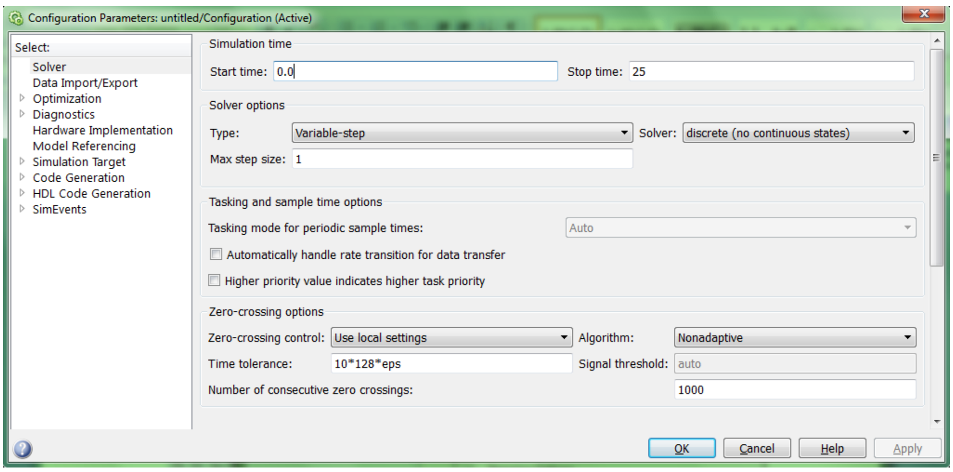


Figura 83 - *Model Configuration Program*

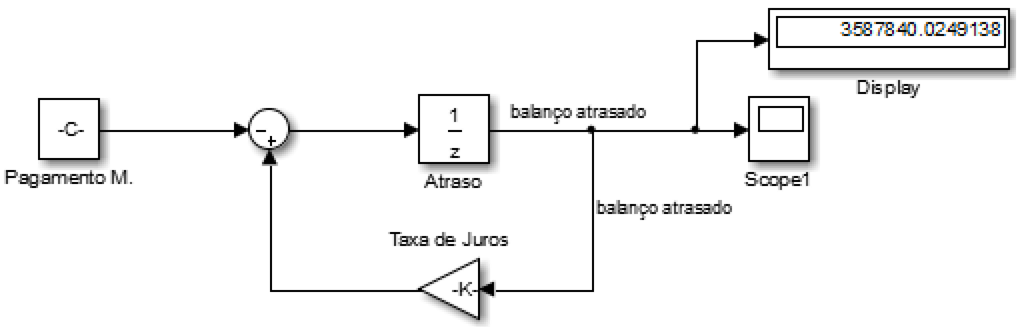


Figura 84 - Pagamento final

O que mostra que ao final dos dois anos, o petiano endividado terá uma dívida de R$ 3.587.840,03 com o banco.

16. GUI

16.1. Introdução e Apresentação

Em computação uma GUI (*Graphical User Interface*) é uma interface gráfica que apresenta um mecanismo mais atraente e mais amigável ao usuário na utilização do software. O MATLAB possui uma ferramenta chamada GUIDE que permite construir interfaces gráficas de interação com o utilizador. O mecanismo auxilia o programador a implementar recursos gráficos mais rápida e facilmente. A Fig. 85 mostra um exemplo de GUI.

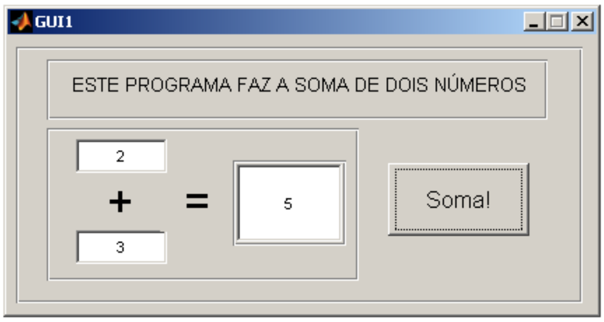


Figura 85 - Exemplo de GUI

Para inicializar a ferramenta GUIDE é necessário digitar *guide* na *Command Window* do MATLAB, Fig. 86, posteriormente clicar em *Blank GUI* (*default*). A tela inicial é mostrada na Fig. 87.

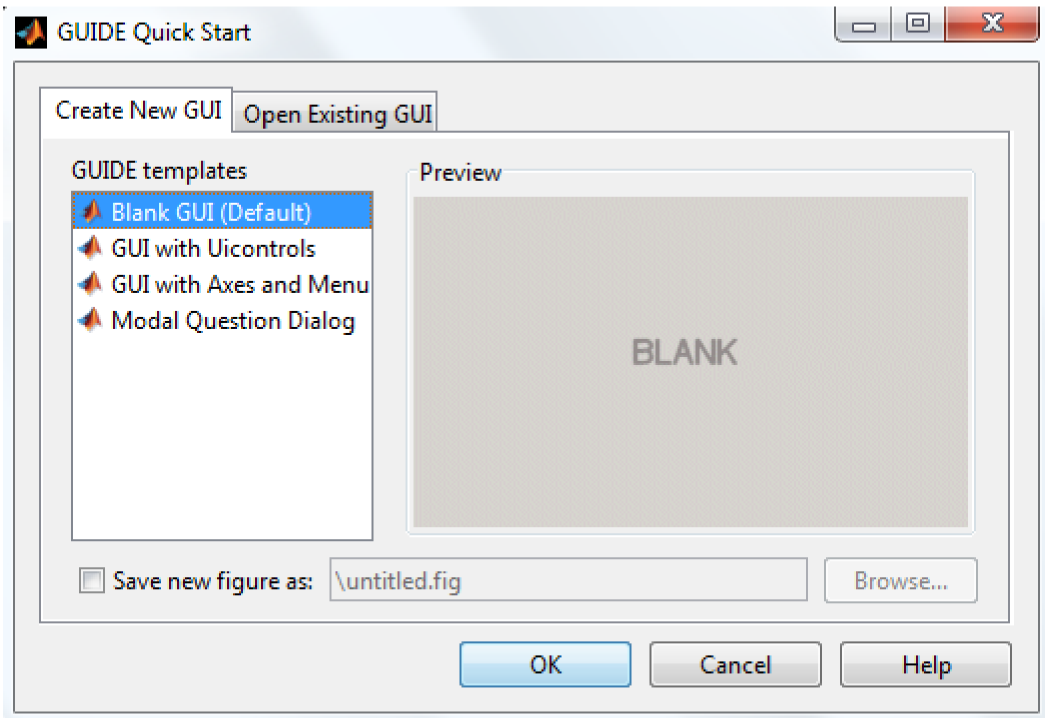


Fig. 86 - GUIDE *Quick Start*

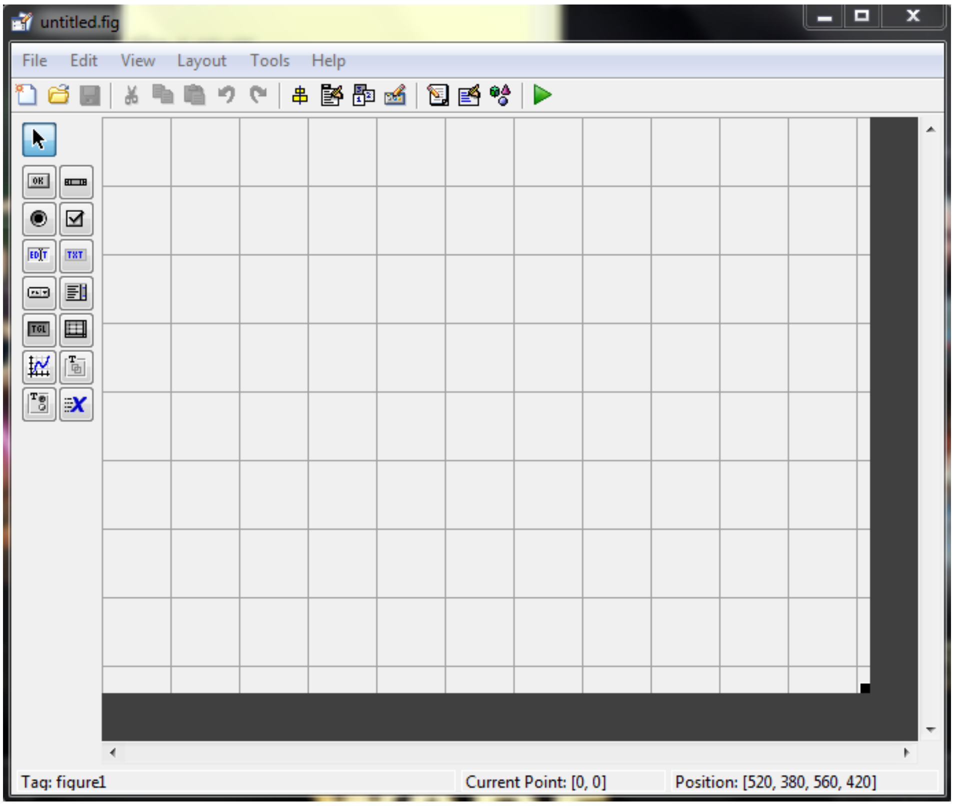


Figura 87 - Tela inicial do GUIDE

Em conjunto com a tela inicial é criado um arquivo .m com o código fonte da GUI criada. Todas as possíveis alterações no modo de operação serão feitas nesse arquivo. O trabalho gráfico será todo feito com o auxílio do mecanismo GUIDE.

16.2. Ferramentas básicas

As ferramentas principais do GUIDE são mostradas na Fig. 88:

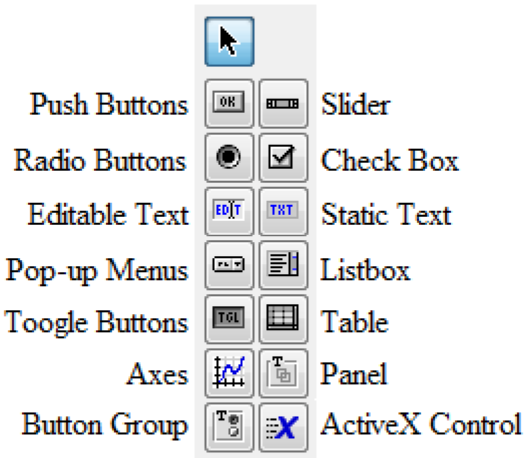


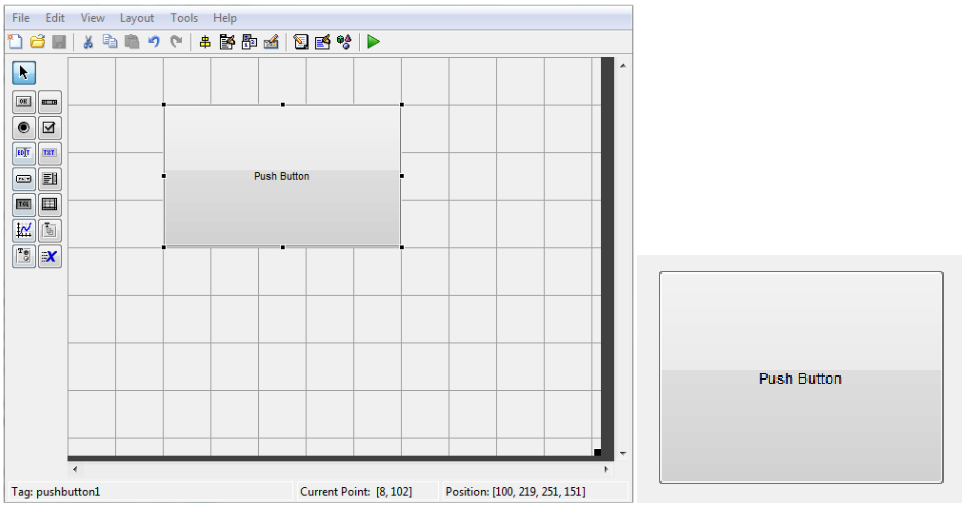
Figura 88 - Ferramentas do GUIDE

De forma mais detalhada, a ferramenta GUIDE tem as seguintes ferramentas:

- ***Push Button***

Com essa ferramenta é possível criar um botão que volta ao estado original após ser ativado. Ao se clicar no ícone *Push Button* da GUI, aparecerá na tela quadriculada uma pequena cruz preta que permite definir o tamanho do botão retangular. Observe na Fig. 89 (a) como fica a criação de um simples botão.

Ao selecionar a função desejada é possível realizar algumas configurações utilizando o *Inspector* sobre a ferramenta como será explicado em seguida. Além disso, o usuário pode fazer diversas combinações de ferramentas, como será mostrado no exemplo do fim do capítulo de GUI. Por enquanto, basta saber que ao executar a ferramenta clicando no ícone verde *Run Figure* aparecerão uma opção para salvar seu trabalho, um arquivo .m do código utilizado para a criação da função e, finalmente, a seguinte tela de interface, Fig. 89 (b).

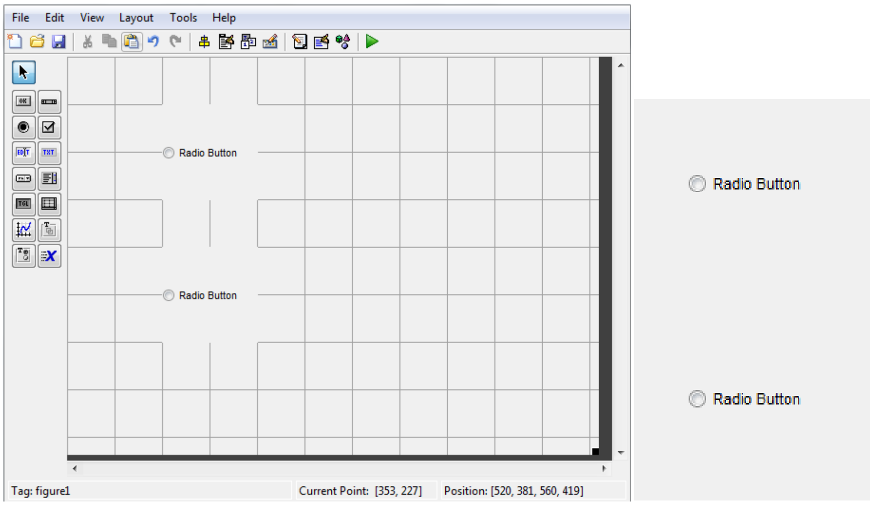


(a) (b)

Figura 89 - *Push Button*; (a) Criação; (b) Interface

- ***Radio Button***

Essa ferramenta permite que o usuário selecione uma opção. Observe a criação de um *Radio Button*, Fig. 90 (a) e sua interface na Fig. 90 (b).

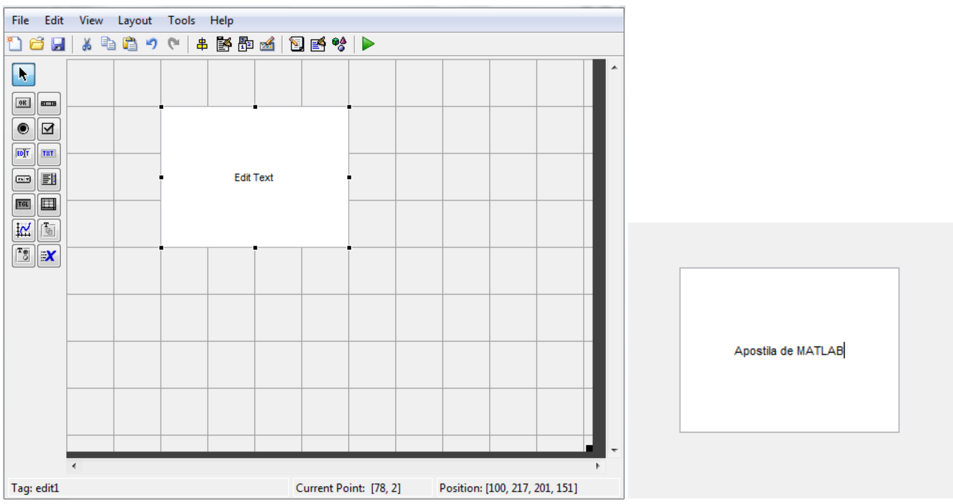


(a) (b)

Figura 90 - *Radio Button*; (a) Criação; (b) Interface

- ***Edit Text***

Campo que recebe textos inseridos pelo usuário. Observe, na Fig. 91 (a) , a inserção de um desses campos. Observe que antes de executar é possível alterar o texto dentro do quadro branco. Em geral, esse quadro é deixado vazio para que o usuário insira um texto. Após a execução, se o quadro tiver sido deixado em branco, aparecerá na interface, Fig. 91 (b), apenas uma pequena tela branca onde o usuário poderá inserir algo.

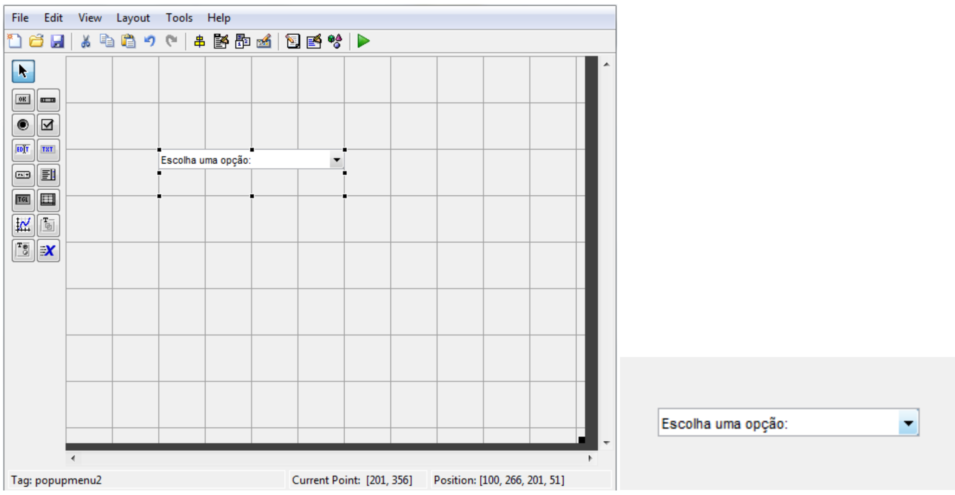


(a) (b)

Figura 91 - *Edit Text*; (a) Criação; (b) Interface

- ***Pop-up* Menu**

Campo que, ao ser clicado, disponibilizada diversas opções criadas pelo programador. A inserção de um *Pop*-up é feita como na Fig. 92 (a) e sua interface está disposta na Fig. 92 (b).

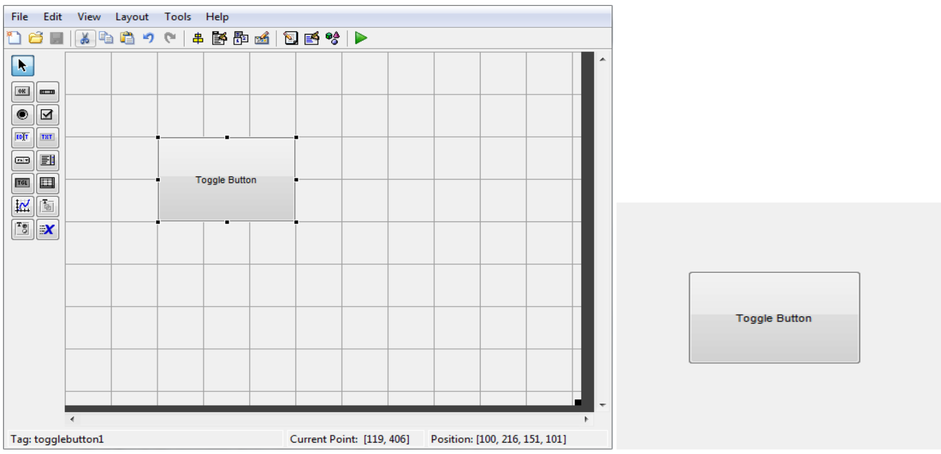


(a) (b)

Figura 92 - *Pop-up* Menu; (a) Criação; (b) Interface

- ***Toogle Buttons***

Semelhante à ferramenta *Push Button*, a *Toggle Button* tem como função exibir um botão a ser ativado. A diferença é que o último permanece pressionado depois que o usuário o ativa. Observe a criação do *Toogle Buttons* na Fig. 93 (a) e sua interface na Fig. 93 (b).

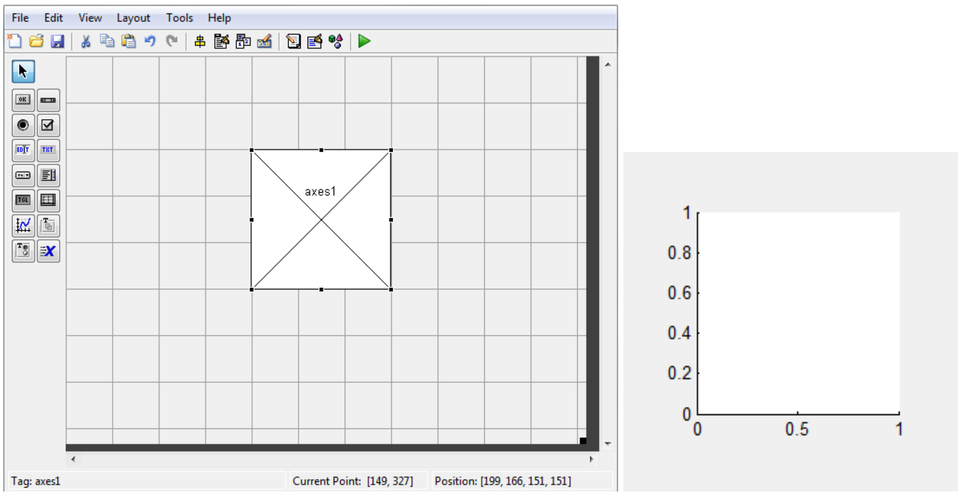


(a) (b)

Figura 93 - *Toogle Buttons;* (a) Criação; (b) Interface

- ***Axes***

Campo que permite a inserção de gráfico na GUI. Observe a criação do *Axes* na Fig. 94 (a) e sua interface na Fig. 94 (b).

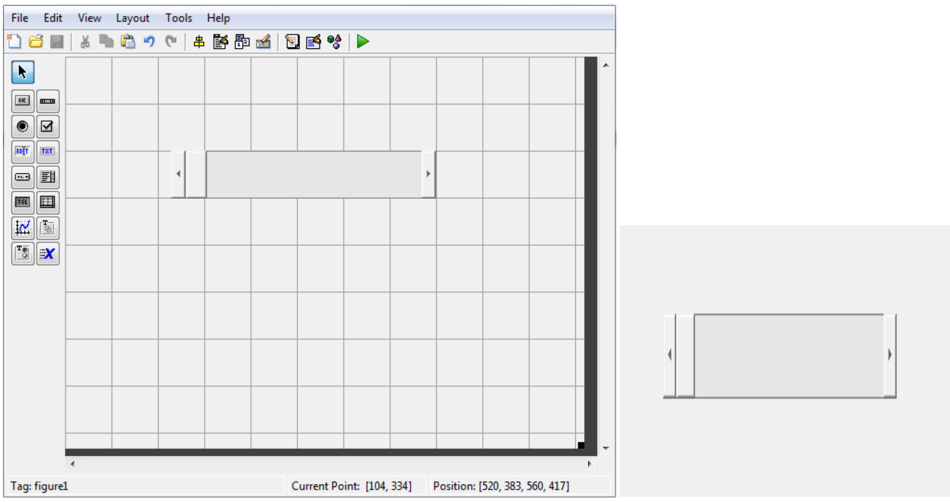


(a) (b)

Figura 94 - *Axes;* (a) Criação; (b) Interface

- ***Slider***

Campo que permite a inserção de uma barra de rolagem. Observe a criação do *Slider* na Fig. 95 (a) e sua interface na Fig. 95 (b).

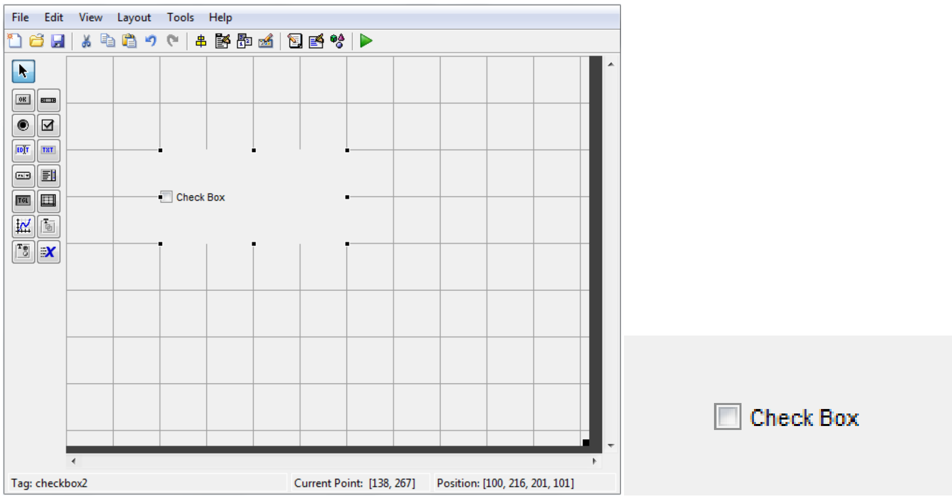


(a) (b)

Figura 95 - *Slider;* (a) Criação; (b) Interface

- ***Check Box***

Botão que permite dentre várias opões escolher quantas o usuário desejar. Observe a criação do *Check Box* na Fig. 96 (a) e sua interface na Fig. 96 (b).

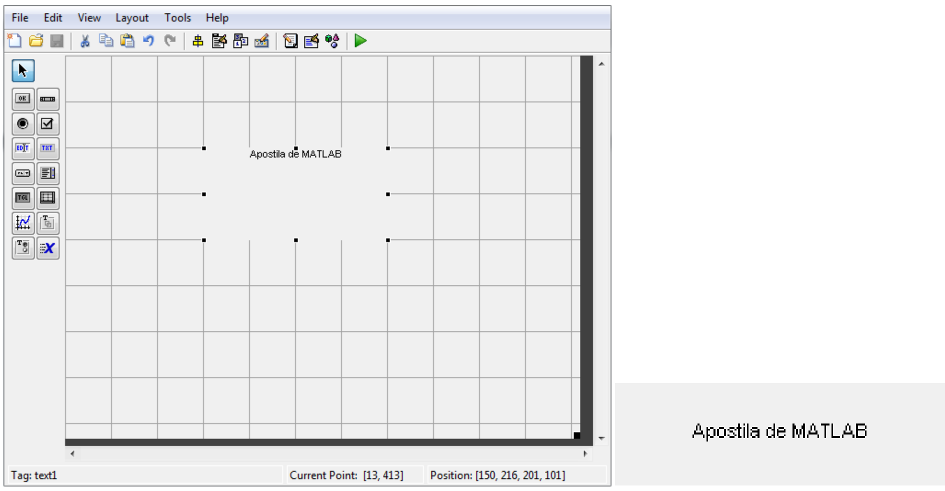


(a) (b)

Figura 96 - *Check Box;* (a) Criação; (b) Interface

- ***Static Text***

Através dessa ferramenta, é possível fazer aparecer um texto que não pode ser modificado pelo usuário. Observe a criação, Fig. 97 (a), e execução, Fig. 97 (b), de uma ferramenta *Static Text*.

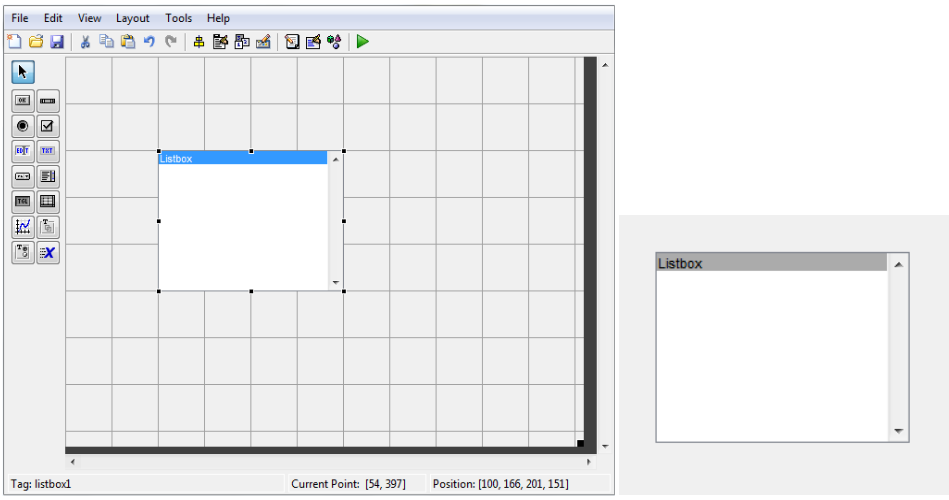


(a) (b)

Figura 97 - *Static Text;* (a) Criação; (b) Interface

- ***Listbox***

Ferramenta que exibe uma lista de itens que o usuário pode selecionar. Observe a criação do *Listbox* na Fig. 98 (a) e sua interface na Fig. 98 (b).

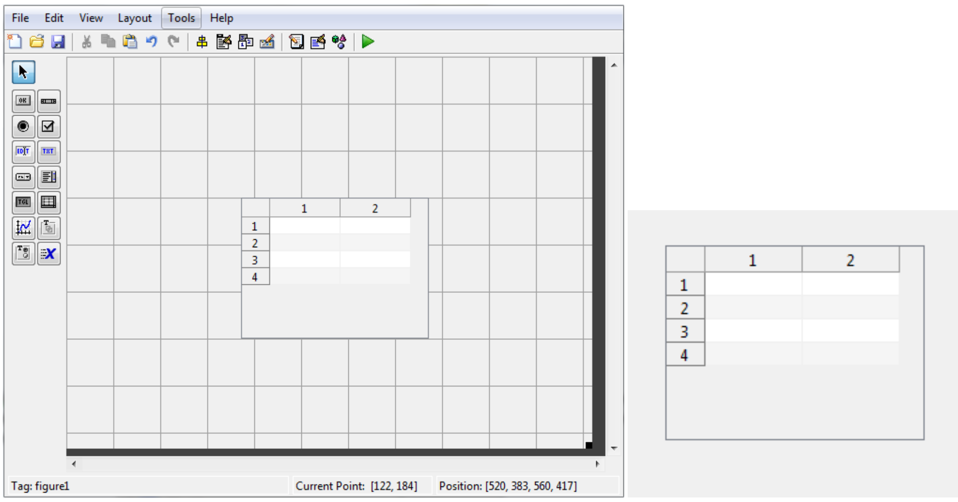


(a) (b)

Figura 98 - *Listbox;* (a) Criação; (b) Interface

- ***Table***

Campo onde são inseridas tabelas na GUI. Observe a criação do *Table* na Fig. 99 (a) e sua interface na Fig. 99 (b).

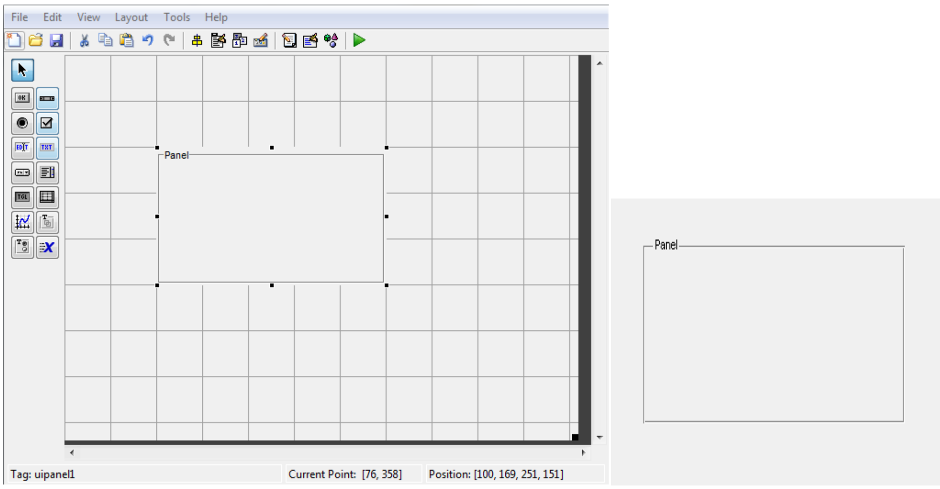


(a) (b)

Figura 99 - *Table;* (a) Criação; (b) Interface

- ***Panel***

Campo que permite o agrupamento de ferramentas quaisquer. A utilidade é organizar as ferramentas. Observe a criação do *Panel* na Fig. 100 (a) e sua interface na Fig. 100 (b).

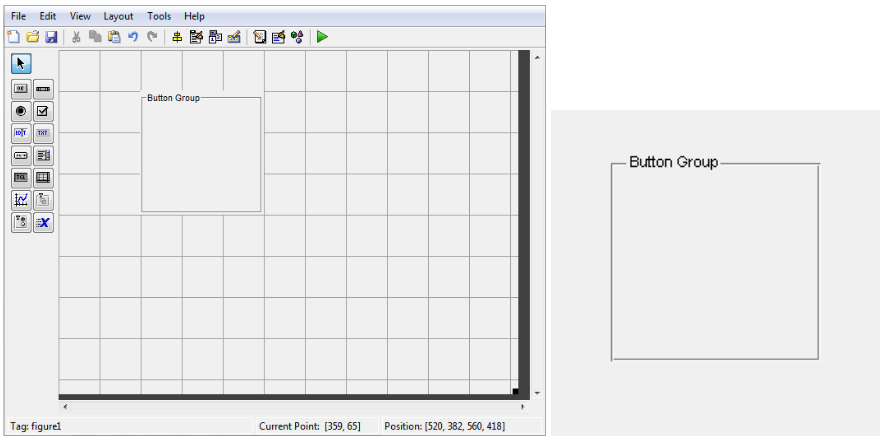


(a) (b)

Figura 100 - *Panel;* (a) Criação; (b) Interface

- ***Button Group***

Campo onde são agrupados botões diversos. Observe que a única diferença dessa ferramenta para um *Panel* é que aqui apenas *Toggle Buttons* e *Radio Buttons* são organizadas. Observe a criação do *Button Group* na Fig. 101 (a) e sua interface na Fig. 101 (b).



(a) (b)

Figura 101 - *Button Group;* (a) Criação; (b) Interface

- ***ActiveX Control***

Permite que os controles da *ActiveX* sejam exibidos na GUI.

Uma vez criados os campos o usuário poderá suas configurar suas propriedades. Para fazê-lo bastar dar um duplo clique em qualquer uma das ferramentas anteriormente apresentadas. A Fig. 102 mostra um exemplo:

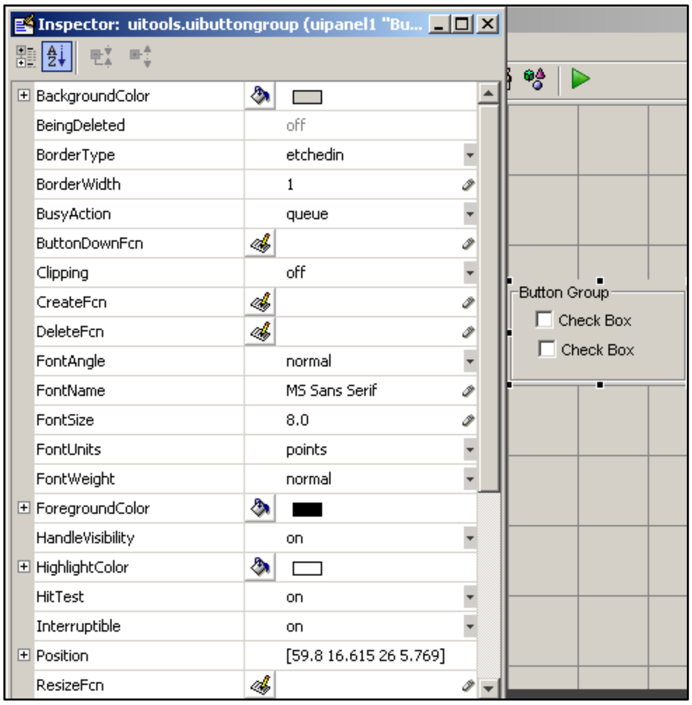


Figura 102 - *ActiveX Control*

Na janela propriedades é possível alterar atributos que vão desde o *design* do objeto até as funcionalidades de dentro da aplicação como um todo.

16.3. Ferramentas *get* e *set*

O GUIDE cria uma função no arquivo .m da aplicação para cada novo objeto adicionado na GUI. Dentre essas funções a mais importante é a *callback*. O *callback* é um função que é escrita e associada a um específico componente da GUI e que controla seu comportamento e suas diversas funções no programa. Dentro da rotina *callback* utilizaremos ainda duas importantes funções:

- ***get***

Definição: Recebe dados inseridos pelo usuário em determinado campo da GUI. Tem várias utilizações sendo a mais utilizada da forma:

Sintaxe:

get(handles. TagDoCampo.'TipoDeVariávelUtizada')

```
function soma_Callback(hObject, eventdata, handles)
A = get(handles.numero1,'String');
B = get(handles.numero2,'String');
A = str2num(A);
B = str2num(B);
C = A+B;
display(C)
```

Interpretação da função apresentada:

Após o botão com *tag* "soma" ser clicado a função acima recebe as *strings* dos campos cujas *tag* são "numero1" e "numero2". É feita a conversão de string para número e ambos são somados. Por fim, o resultado é mostrado na tela, Fig. 103.

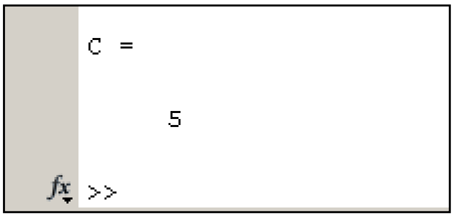


Figura 103 - Tela de resultado do comando *get*

- ***set***

Definição: Envia dados que são editáveis para determinado campo da GUI. É a função complementar ao *get* e também tem várias utilizações sendo a mais utilizada da forma:

Sintaxe:

get(handles.TagDoCampo.'TipoDeVariávelUtizada',InformaçãoEnviada)

```
function botao_Callback(hObject, eventdata, handles)

display('Aprendi a fazer um SET no GUIDE!!!')
set(handles.texto,'string', 'Aprendi a fazer um SET no GUIDE!!!')
```

Interpretação da função apresentada:

Após o botão com *tag* "botão" ser clicado a função acima recebe/ envia uma mensagem do tipo *string* para um objeto com *tag* "texto". Caso o objeto seja um *Editable Text* a tela da GUI será similar a da Fig. 104.

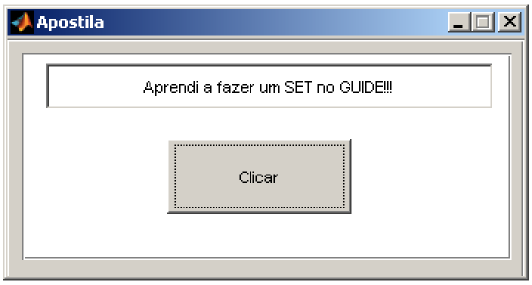


Figura 104 - Tela de resultado do comando *set*

16.4. Criando uma GUI

1. Abrir um arquivo em branco no GUIDE, Fig. 105.

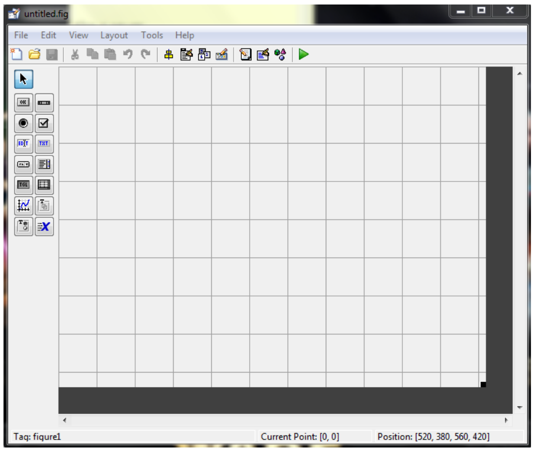


Figura 105 - Passo 1 de criação da GUI

2. Inserir os componentes desejados, Fig. 106.

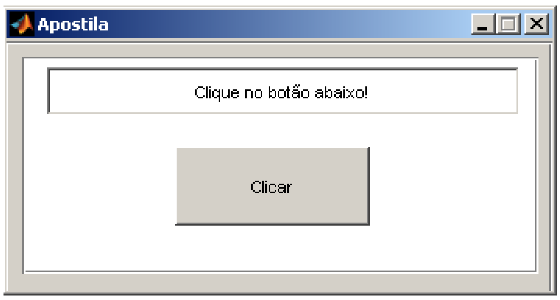


Figura 106 - Passo 2 de criação da GUI

3. Configure os atributos, Fig. 107

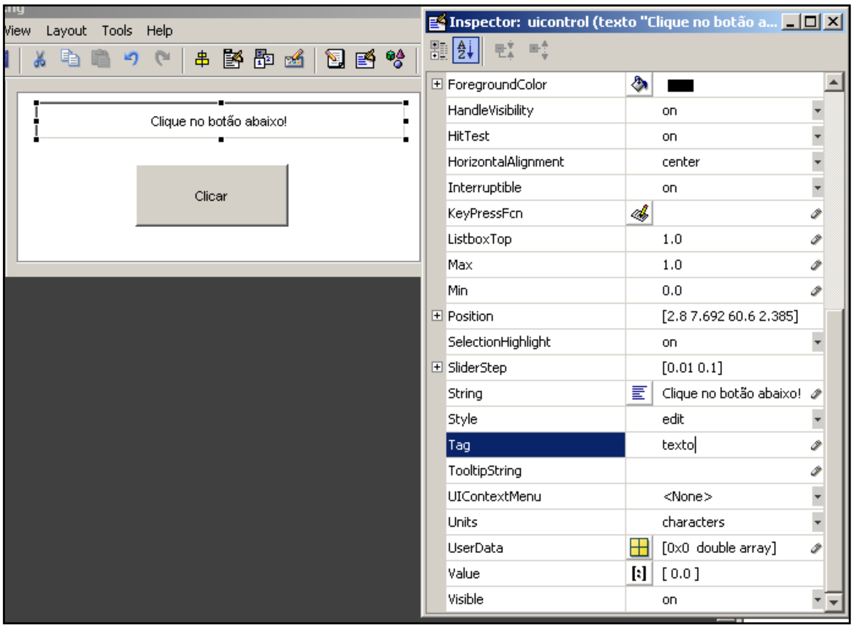


Figura 107 - Passo 3 de criação da GUI

4. Manipule o m-file.

```
   function botao_Callback(hObject, eventdata, handles)

   display('Aprendi a fazer um SET no GUIDE!!!')
   set(handles.texto,'string', 'Criei minha primeira GUI no MATLAB!!!
Uhhhuuuu')
```

5. Compile e aproveite, Fig. 108:



Figura 108 - Passo 5 de criação da GUI

Exercício 27 - Crie uma GUI que funcione como calculadora científica de acordo com a Fig. 109:

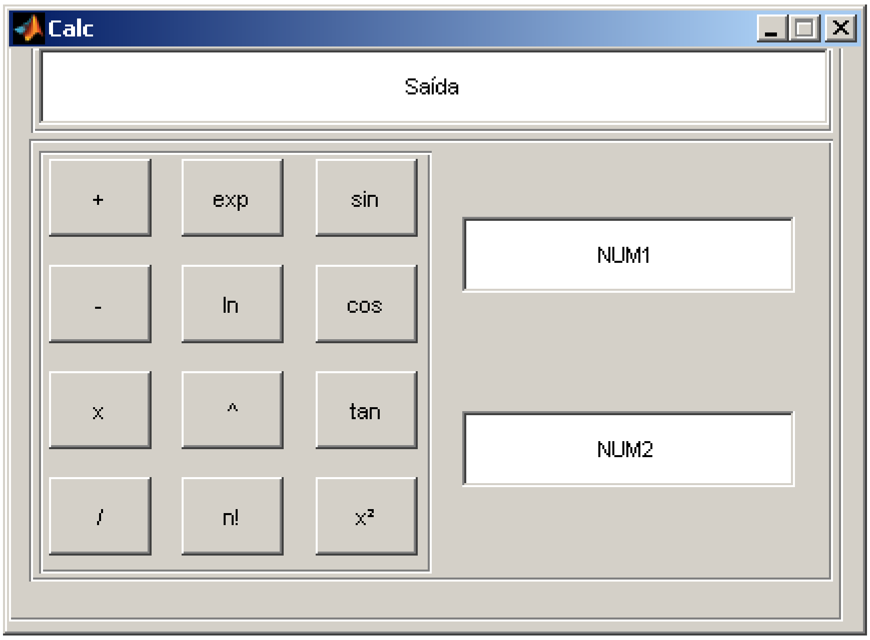


Figura 109 - Exercício proposto

17. REFERÊNCIAS BIBLIOGRÁFICAS

[1] **CARNAHAN**, Brice, **LUTHER**, H. A. & **WILKES**, James O. *Applied numerical methods*. John Wiley & Sons, Inc. Nova Iorque – 1969.

[2] **GÓES**, Hilder & **TONAR**, Ubaldo. *Matemática para Concursos*. 6ª Edição. ABC Editora. Fortaleza – CE. 2001.

[3] **HAYKIN**, Simon & **VEEN**, Barry Van. *Sinais e Sistemas*. Editora Bookman. Porto Alegre – RS. 2001

[4] **HAYT**, William H. Jr. & **BUCK**, Jonh A. *Eletromagnetismo*. 6ª Edição. Editora LTC. Rio de Janeiro – RJ. 2001.

[5] **LEITHOLD**, Louis. *O Cálculo com Geometria Analítica*. 3ª Edição. Volume I. Editora Habra. São Paulo – SP. 1994.

[6] **NILSSON**, James W & **RIEDEL**, Susan A. *Circuitos Elétricos*. 6ª Edição. Editora LTC. Rio de Janeiro – RJ. 2003.